P. 1847/90

11-12 1990

informatyka

Najlepsze życzenia
świąteczne i noworoczne
wszystkim Czytelnikom i Autorom
składa
zespół redakcyjny

WYDAWNICTWO NOT

Cena 11 600 zł

informatyka

nr 11–12 1990
Listopad–grudzień
Rok wydania XXV

Miesięcznik

ISSN 0542-9951. INDEKS 36124

KOLEGIUM REDAKCYJNE:

mgr Jarosław DEMINET,
dr inż. Wacław ISZKOWSKI,
mgr Teresa JABŁOŃSKA,
(sekretarz redakcji)
Władysław KLEPACZ
(redaktor naczelny)
dr inż. Wojciech MOKRZYCKI,
mgr inż. Jan RYŻKO,
mgr Hanna WŁODARSKA.

PRZEWODNICZĄCY
RADY PROGRAMOWEJ:

Prof. dr hab.
Juliusz Lech KULIKOWSKI

WYDAWCA: Wydawnictwo Czasopism i Książek Technicznych SIGMA NOT
Spółka z o.o.
ul. Biała 4,
00-950 WARSZAWA
skrytka pocztowa 1004

Redakcja: 01-552 Warszawa,
Pl. Inwalidów 10 p. 104, 105
tel. 39-14-34

Materiałów nie zamówionych redakcja nie zwraca

DRUK:
PRASOWE ZAKŁADY GRAFICZNE
RSW „PRASA-KSIĄŻKA-RUCH"
ul. Dworcowa 13,
85-950 BYDGOSZCZ
zam. 2790/90
Obj. 6,5 ark. druk. Nakład 5100 egz

Cena egzemplarza podwójnego 11 600 zł

W sprawach ogłoszeń prosimy zwracać się bezpośrednio do Redakcji

W numerze:

	Strona
Systemy CASE – problemy, techniki, rozwiązania (1) – *Piotr Fuglewicz*	1
Przechowywanie i przetwarzanie danych o nieregularnych strukturach – *Krzysztof Dworakowski, Henryk Klimek, Elżbieta Przepióra*	2
Medytor – mówiący edytor języka polskiego – *Mieczysław Kłopotek*	13
Język Lisp i elementy sztucznej inteligencji (2). Konstruowanie funkcji i programu – *Zdzisław Wrzeszcz*	16
Fizyczne struktury systemów baz danych (3). Wielowymiarowe struktury danych – *Miłosz Muszyński*	21
Systemy ekspertowe w projektowaniu układów analogowych – *Henryk Budzisz*	27
Poczta elektroniczna systemu Advanced Netware 286 – *Katarzyna Lange*	32
Z kraju	
III Walny Zjazd PTI – *Jarosław Deminet*	36
/unix	
Co po czym i kiedy (o ewaluacji wejścia w shellu Bourne'a) – *Jan Walasek*	38
Systemy otwarte i ich wrogowie, czyli co dalej z UNIX-em (3). Czy grozi nam standardowy UNIX? – *Jacek Surma*	40
Czy mógłbyś mnie zabić? (czyli o utracie kontaktu z rzeczywistością) – *Dariusz Kupiecki*	43
Argumenty, parametry, środowisko (w shellu Bourne'a) – *Jan Walasek*	44
Mimika pocztowa – *JS*	46
The Design of the Unix Operating System – *Jacek Surma*	46
zBITki	
LET DRENAŻ = Pomoc – *Wacław Iszkowski*	III okł.

W najbliższych numerach:

- Juliusz Lech Kulikowski opisuje komputerowe przetwarzanie obrazów dla potrzeb diagnostyki lekarskiej i badań biologicznych.
- Wojciech Skaba omawia współczesne metody kompresji danych: teorię, algorytmy i zastosowania.
- Piotr Fuglewicz, w drugiej części artykułu na temat wspomaganej komputerowo inżynierii oprogramowania CASE, charakteryzuje sześć opracowanych w kraju programów z tej dziedziny.

Warunki prenumeraty

Przyjęcie prenumeraty – wyłącznie na podstawie dokonanej wpłaty na drukach dostarczanych dotychczasowym prenumeratorom przez Wydawnictwo, lub nowym – po uprzednim zgłoszeniu zapotrzebowania (pisemnie lub telefonicznie) w Zakładzie Kolportażu Wydawnictwa.

Blankiet wpłaty – powinien zawierać następujące informacje: dokładna nazwa i adres (z kodem pocztowym) zamawiającego, tytuły zamawianych czasopism, ich liczbę i okres prenumeraty.

Wpłata – zgodnie z podanymi cenami należy dokonać w banku lub w UPT na konto podane na naszym blankiecie, tj.:
Państwowy Bank Kredytowy III O/Warszawa nr: 370015-1573-139-11

Prenumeratorzy zbiorowi – osoby prawne obowiązują blankiety „Wpłata-Zamówienie". Cena normalna.

Prenumeratorzy indywidualni – osoby fizyczne obowiązują blankiety typu przekazy dla wpłat na rachunki bankowe. Cena normalna.

Prenumerata ulgowa – zgodnie z podaną ceną ulgową przysługuje wyłącznie osobom fizycznym, będącym członkami SNT, studentom i uczniom szkół zawodowych. Uczniowie szkół ogólnokształcących mogą zamówić w prenumeracie ulgowej tylko miesięcznik „Aura".
Uwaga: w podanym okresie prenumeraty można zamówić tylko po jednym egzemplarzu z każdego tytułu.

Prenumerata ze zleceniem wysyłki za granicę – cena prenumeraty jest dwukrotnie wyższa od ceny normalnej.
Należy podać dokładny adres odbiorcy za granicą.

Terminy przyjmowania prenumeraty:
do 10 listopada na I, II, III, IV kwartał następnego roku
do 28 lutego na II, III, IV kwartał br.
do 31 maja na III, IV kwartał br.
do 31 sierpnia na IV kwartał br.
Zmiany w prenumeracie, np. zmiana liczby tytułów, liczby egzemplarzy, rezygnacja z prenumeraty, można zgłaszać tylko w podanych terminach z mocą obowiązującą od następnego kwartału.

Egzemplarze archiwalne (z lat ubiegłych)
Można nabyć za gotówkę w Klubie Prasy Technicznej w Warszawie, ul. Mazowiecka 12 (tel. 26-80-16) lub zamówić pisemnie w Zakładzie Kolportażu, Dział Handlowy, 00-950 Warszawa skr. poczt. 1004 (tel. 40-37-31) na rachunek lub za zaliczeniem pocztowym.

Informacji o prenumeracie udziela: Zakład Kolportażu Wydawnictwa SIGMA-NOT Spółka z o.o., 00-716 Warszawa, ul. Bartycka 20, skr. 1004. Telefony: 40-00-21 wewn. 293, 295, 299 lub 40-30-86, 40-35-89.

Wstępna cena jednego egzemplarza na 1990 rok: normalna – 5800 zł
ulgowa – 2900 zł

Wartość prenumeraty (w zł):
Normalna: kwartalna – 17 400, półroczna 34 800, roczna 69 600
Ulgowa: kwartalna – 8700, półroczna 17 400, roczna 34 800

Uwaga: W przypadku zmiany cen w okresie objętym prenumeratą, prenumeratorzy zobowiązani są do dopłaty różnicy cen.

P. 1877/90

POLITECHNIKA ŚLĄSKA Biblioteka Główna Gliwice

XXV lat INFORMATYKI

W końcu listopada br. minęło 25 lat od ukazania się naszego miesięcznika na rynku wydawniczym. Ćwierćwiecze jest jubileuszem szczególnie uzasadnionym, ponieważ stanowi równoważnik powszechnie przyjętej miary jednego ludzkiego pokolenia. Zgodnie z obowiązującym zwyczajem mamy więc obowiązek nie tylko formalnego odnotowania tego faktu, ale również przypomnienia niektórych szczegółów drogi rozwoju czasopisma.

Przypomnienia, a właściwie odtworzenia, wymaga zwłaszcza geneza powstania oraz początkowy okres działalności, co nie jest łatwe ze względu na brak jakiejkolwiek dokumentacji. Źródłem informacji mogła być jedynie pamięć członków zespołu założycielskiego, ale i tu okazuje się, że znaczny upływ czasu wyklucza już możliwość dotarcia do wszystkich bezpośrednich uczestników. Ponieważ INFORMATYKA od 1965 r. była z tytułu swych funkcji znaczącym stymulatorem rozwoju polskiej informatyki, informacje te warto utrwalić i tym samym uchronić od zapomnienia w podobny sposób, jak to nastąpiło rok temu w odniesieniu do pierwszych dwudziestu lat rozwoju tej dziedziny (materiały okolicznościowej konferencji PTI, INFORMATYKA nr 8–12, 1989).

Pomysł stworzenia krajowego czasopisma naukowo-technicznego poświęconego komputerom (nazywanych początkowo maszynami matematycznymi lub elektronicznymi maszynami cyfrowymi) oraz ich zastosowaniom wysunął na początku 1965 roku prof. Leon Łukaszewicz, dyrektor Instytutu Maszyn Matematycznych PAN, wiodącego w tym czasie ośrodka badawczego w dziedzinie konstrukcji, programowania i zastosowań komputerów. Realizację tego pomysłu prof. Łukaszewicz zlecił inż. Dorocie Prawdzic, kierownikowi branżowego ośrodka informacji naukowo-technicznej w tymże Instytucie. Okazało się wówczas, że podstawową trudnością było znalezienie wydawcy, gotowego podjąć ryzyko dość wątpliwego, jak oceniano, przedsięwzięcia. Spośród wszystkich znanych oficyn wydawniczych, do których się zwrócono, jedynie Wydawnictwo Czasopism Technicznych NOT wykazało odwagę. Decyzję tę podjął ówczesny, dziś nie żyjący już naczelny dyrektor WCT NOT, inż. Wacław Czarnowski. Mimo przedemerytalnego wieku, dyr. Czarnowski wykazał wtedy prawdziwie młodzieńczą wyobraźnię i pełne zrozumienie znaczenia nowej dziedziny techniki, która była przecież domeną wyłącznie ludzi bardzo młodych. Dziedziny, która nie miała jeszcze poparcia u ówczesnych decydentów, a ponadto budziła wątpliwości również w szerokich kręgach społeczeństwa. Pionierski charakter decyzji powołania pisma potwierdza dodatkowo fakt, że polska inicjatywa wyprzedziła analogiczne działania we wszystkich krajach RWPG (istniejące już podobne czasopisma w NRD oraz Czechosłowacji poświęcone były prawie wyłącznie elektromechanicznym maszynom do przetwarzania danych). Oceniając rozmiar podejmowanego przez dyr. Czarnowskiego ryzyka wydawniczego należy uświadomić sobie bardzo ograniczony wówczas krąg odbiorców proponowanego czasopisma, obejmujący jedynie kilka placówek badawczych i wyższych uczelni oraz garstkę entuzjastów-amatorów zastosowania nowej techniki. Przyjęty na początek stosunkowo duży – w porównaniu do innych czasopism technicznych – nakład 2000 egzemplarzy okazał się trafny i już wkrótce uległ zwiększeniu.

Znacznie łatwiejszym zadaniem organizacyjnym było skompletowanie zespołu redakcyjnego. Istniała realna możliwość wykorzystania autentycznego entuzjazmu i bezinteresownego zaangażowania, zjawisk występujących wtedy w znacznej skali, a charakterystycznych dla wyjątkowo młodego wieku ówczesnych specjalistów. Podstawowym kryterium selekcji kandydatów na redaktorów spośród wielu zainteresowanych były – oprócz osiągniętej pozycji zawodowej i potrzebnego kierunku specjalizacji informatycznej – niezbędne doświadczenie publikacyjne. Poza wspomnianymi już organizatorami, w skład pierwszego zespołu kolegium redakcyjnego weszły ostatecznie następujące osoby: Konrad Fiałkowski, Wojciech Jaworski, Władysław Klepacz, Tomasz Pietrzykowski oraz Andrzej Targowski. Funkcje redaktora naczelnego objął prof. L. Łukaszewicz, natomiast pozostali członkowie kolegium otrzymali tytuły redaktorów działu.

Formalne umowy zlecenia, mające w wymiarze finansowym charakter wyłącznie symboliczny, członkowie kolegium redakcyjnego otrzymali 1 lipca 1965 r. Przygotowanie materiałów do druku nastąpiło w ciągu dwóch wakacyjnych miesięcy i po typowym trzymiesięcznym cyklu poligraficznym, pierwszy numer dwumiesięcznika pod niezbyt atrakcyjnym tytułem MASZYNY MATEMATYCZNE dotarł do odbiorców pod koniec listopada 1965 r.

O tym, że przydzielone nowemu czasopismu przez Wydawcę środki finansowe były bardziej niż symboliczne najlepiej świadczy fakt zatrudnienia etatowego sekretarza redakcji dopiero po upływie pełnego roku. Do tego momentu cały olbrzymi ciężar organizowania i prowadzenia procesu wydawniczego przyjęła na swe barki red. Prawdzic, która po niespełna 10 latach bardzo intensywnej działalności redaktorskiej, przedwcześnie zmarła. Z pozostałych członków zespołu założycielskiego czterech redaktorów przed wielu laty opuściło kraj (K. Fiałkowski, W. Jaworski, T. Pietrzykowski i A. Targowski), a prof. Łukaszewicz, po 20 latach nieprzerwanej działalności, wycofał się z obowiązków kierowania redakcją. W ten sposób jedynym członkiem zespołu założycielskiego, jaki pozostał w redakcji po dzień dzisiejszy jest aktualny naczelny redaktor INFORMATYKI.

Niewielu z naszych obecnych czytelników wie o tym, że obecna nazwa czasopisma została wprowadzona od 1971 roku z inspiracji red. A. Targowskiego, który wylansował i niespodziewanie szybko upowszechnił w naszym kraju powstały we Francji termin „informatyka". Do jego upowszechnienia w znacznej mierze przyczynił się niewątpliwie również nasz tytuł. Należy przy tym zauważyć, że to bardzo trafne i przyjemne dla ucha słowo w ostatnich latach rozpowszechniło się także w obszarze języków angielskiego oraz niemieckiego, ale Polacy uczynili to około 10 lat wcześniej!

Od samego początku profil czasopisma został jednoznacznie ukierunkowany na zastosowania komputerów, co potwierdzał istniejący przez pełne 20 lat jego podtytuł „Zastosowania w gospodarce, technice i nauce". Ukierunkowanie takie, zmodyfikowane dopiero w latach osiemdziesiątych, zapewniało nie tylko stały wzrost kręgu czytelników, którzy w znacznej mierze rekrutowali się spośród przyszłych użytkowników narzędzi informatycznych, ale również było podstawowym stymulatorem rozwoju całej branży, a w konsekwencji sukcesywnego wzrostu nakładu INFORMATYKI (w roku ubiegłym zbliżył się on do granicy 10 tys. egzemplarzy).

Na krajowym rynku wydawniczym INFORMATYKA utrzymuje w kategorii periodyków profesjonalnych nadal pozycję monopolisty, co jest swoistym wyrazem ułomności tego rynku. Obserwowana obecnie łatwość i znaczna szybkość realizacji inicjatyw wydawniczych pozwala przypuszczać, że w niedalekiej już przyszłości pojawią się w kraju inne, wyspecjalizowane periodyki informatyczne, upodobniając nas wreszcie do sytuacji, jaka występuje od wielu lat w krajach gospodarczo rozwiniętych.

Na koniec refleksja. Stwierdzenie końcowe artykułu okolicznościowego opublikowanego w grudniowym numerze 1985 r. z okazji XX lat INFORMATYKI, które formułowało nadzieję, że Polska za lat pięć wyrwie się z informatycznego zapóźnienia i odzyska swoją pierwotną pozycję prekursora w naszym rejonie Europy, niestety jeszcze nie spełniło się. Widoczne są jednak wyraźne przesłanki tego, że może to nastąpić znacznie wcześniej, niż dziś przewidują zawodowi prognostycy.

Władysław Klepacz

UWAGA UŻYTKOWNICY SYSTEMÓW KOMPUTEROWYCH **PRACUJĄCYCH POD KONTROLĄ S.O. UNIX!**

PRZEDSIĘBIORSTWO TECHNICZNO-HANDLOWE
„KaNet" Sp. z o.o.
41–500 CHORZÓW, ul. Katowicka 77
tel. 412-122, teleks 312627 kanet pl

POLECA:

- **NOWOCZESNE ROZWIĄZANIA SPRZĘTOWE FIRMY TEXAS INSTRUMENTS (system operacyjny UNIX),**
- **PROGRAMY NARZĘDZIOWE PAKIETU AUTORSKIEGO mixPACK:**
  - * mixCOMM: program wspomagający zarządzanie plikami (wyróżnienie oraz nagroda Ministra Przemysłu na Ogólnopolskich Targach Oprogramowania Softarg '90)
  - * mixEDIT: pełnoekranowy edytor tekstowy,
  - * mixOFFI: pakiet wspomagający prace biurowe,
- **OPROGRAMOWANIE APLIKACYJNE Z ZAKRESU ORGANIZACJI I ZARZĄDZANIA PRZEDSIĘBIORSTWEM (pakiet orgPACK),**
- **SYSTEMY UŻYTKOWE REALIZOWANE WG INDYWIDUALNYCH POTRZEB UŻYTKOWNIKA.**

„KaNet" POSIADA PRAWA AUTORSKIE NA CAŁOŚĆ OFEROWANEGO OPROGRAMOWANIA!
„KaNet" GWARANTUJE WIELOLETNIE UŻYTKOWANIE ZAKUPIONEGO OPROGRAMOWANIA DZIĘKI MOŻLIWOŚCI PRZENOSZENIA GO NA DOWOLNE SYSTEMY KOMPUTEROWE!

ZAPRASZAMY DO KORZYSTANIA Z NASZEJ OFERTY!

0/16/90

Pierwsza wersja tego artykułu została przedstawiona podczas Trzeciej Wiosennej Szkoły PTI pn. „Komputerowe wspomaganie tworzenia systemów informatycznych", która odbyła się w dniach 14–18 maja 1990 r. w Świnoujściu.

POLSKIE TOWARZYSTWO INFORMATYCZNE

PIOTR FUGLEWICZ
Katowice

Systemy CASE problemy, techniki, rozwiązania (1)

Wspomagana komputerowo inżynieria oprogramowania (ang. CASE – *Computer Aided Software Engeneering*) jest pewnego rodzaju hasłem reklamowym, pozwalającym sprzedawać bardzo różne produkty informatyczne. Takimi hasłami były w historii informatyki: strukturalność, społeczliwość (ang. *human friendliness*) i inne powtarzające się w ogłoszeniach reklamowych cechy proponowanego oprogramowania. CASE jest również synonimem automatyzacji tworzenia programów z zastosowaniem metod i technik, będących dorobkiem dotychczasowej praktyki wykonywania systemów informatycznych.

Zakres komputerowego wspomagania inżynierii programowania obejmuje:
- **identyfikację obiektu, dla którego należy skonstruować system informatyczny,**
- **analizę i dekompozycję problemu na składowe odpowiadające elementom tego systemu,**
- **dobór najwłaściwszych narzędzi i metod realizacji tych składowych,**
- **syntezę systemu informatycznego.**

Od samego początku, a w każdym razie od chwili opracowania pierwszego asemblera, praca programistów była wspomagana komputerowo. Do tworzenia programów użytkowych używano zawsze oprogramowania narzędziowego. W latach sześćdziesiątych i pierwszej połowie lat siedemdziesiątych oprogramowanie narzędziowe koncentrowało się przede wszystkim na etapie kodowania programu. Był to okres gwałtownego rozwoju języków programowania – zaniedbywano rozwój narzędzi wspomagających inne fazy cyklu życia produktu programowego. Obecnie dzięki rozwojowi inżynierii programowania, opracowano wiele różnorodnych narzędzi wspomagających wszystkie fazy cyklu życia produktu programowego. Niestety, nie istnieje jeden, ogólnie przyjęty model cyklu życia produktu. Jednym z używanych jest model kaskadowy, w którym cykl życia projektu programowego składa się z następujących faz [30]:

- specyfikacja założeń,
- projekt struktury,
- projekt szczegółowy,
- kodowanie,
- testowanie i weryfikacja,
- integracja,
- pielęgnowanie.

Model ten dość naturalnie odpowiada działaniom informatyka tworzącego produkt programistyczny. Należy jednak zwrócić uwagę, że rzeczywista praca nie przebiega prawie nigdy w sposób sekwencyjny i że praktycznie z każdej fazy cyklu może nastąpić powrót do dowolnej z poprzednich. To spostrzeżenie leży u podstaw filozofii CASE, będącej próbą stworzenia spójnej metody identyfikacji obiektów przez wykorzystanie komputera jako narzędzia ułatwiającego opisywanie ich struktury oraz odpowiedników tej struktury wyrażonej w kategoriach informatycznych.

Podstawową sprawą w CASE są techniki i metodologie. Istnieje co najmniej kilkanaście metod opisu struktur obiektów i odpowiadających im systemów informatycznych. Jedne są wsparte mocną teorią matematyczną, jak np. sieci Petriego, inne bardziej intuicyjne, jak podejście Yourdona. Nie ma jednak w CASE, jak na razie, żadnych standardów. Producenci firmują określeniem CASE każdy produkt, który w dowolny sposób automatyzuje prace nad oprogramowaniem. Ambitne zadania określenia zasad i norm stawiają przed sobą między innymi: Software Engeneering Institute w Carnegie-Mellon University oraz zwłaszcza CAIM (Center for Advanced Information Menagement – Centrum Zaawansowanego Zarządzania Informacją). CAIM jest niedochodową organizacją afiliowaną przy Uniwersytecie w Auburn, w skład której wchodzą członkowie zarówno ze środowiska akademickiego, jak i z przemysłu. Celem jej istnienia są badania w dziedzinie inżynierii oprogramowania. Zadanie CAIM polega na stworzeniu forum dla prac badawczych nad potrzebami i metodami ich zaspokajania w dziedzinie szeroko rozumianego przetwarzania informacji. Głównymi obszarami zainteresowania tej instytucji są:

- planowanie organizacyjne,
- analiza i projektowanie systemów zarządzania,
- rozwój i utrzymanie systemów informatycznych,
- standaryzacja dokumentacji,
- systemy czasu rzeczywistego,
- zarządzanie zasobami informacyjnymi,
- wspomaganie projektowania.

Długofalowym celem działania CAIM jest sformułowanie międzynarodowych standardów dla różnych aspektów inżynierii oprogramowania. Jego realizacji ma służyć powołanie czasopisma *Journal of Information Engeneering*, które ma się stać platformą wymiany informacji między zainteresowanymi tą problematyką. Centrum ma zamiar ściśle współdziałać z ISO (*International Standarts Organization*) oraz USASI (United States of America Standards Institute) w określaniu standardów dla różnych gałęzi inżynierii oprogramowania.

Narzędzia CASE

Do celów informatyzacji zarządzania powstają liczne programy narzędziowe typu CASE. Można wśród nich wyróżnić dwie klasy:

- pakiety narzędziowe (ang. *toolkits*),
- pakiety zintegrowane o charakterze zamkniętym (ang. *workbenches*).

Pakiety narzędziowe tworzone w jednej firmie, jeśli stają się w jakimś zakresie standardem, są rozszerzane przez dodatkowe programy, dostępne u wielu dostawców. Użytkownik musi wybierać między jednolitością stosowanych narzędzi, czemu sprzyja ich zakup u jednego producenta, a różnorodnością udostępnianych środków, czemu z kolei sprzyja urozmaicenie źródeł dostaw. Zintegrowane pakiety CASE zawierają z reguły następujące środki programowe:

- narzędzia specyfikacji i interpretacji opisu systemu,
- generatory struktur baz danych,
- generatory programów wykonawczych,
- programy dokonujące modyfikacji wersji systemu.

Zarówno w programach narzędziowych, jak i pakietach CASE stosuje się powszechnie graficzne środki wymiany informacji z użytkownikiem. Większość metodologii stosowanych w CASE oparto na graficznej prezentacji właściwości projektowanego systemu informatycznego. W systemach zintegrowanych jądrem, wokół którego organizuje się cały system, jest swoista baza danych, nazywana słownikiem danych (ang. *data dictionary*) lub słownikiem opisu aplikacji. W takiej bazie przechowuje się informacje tekstowe zarówno dotyczące nazw elementów danych i struktury systemu, jak i informacje czysto opisowe.

Zależnie od rozmiarów docelowego systemu informatycznego, do jego wyprodukowania stosuje się komputery różnej mocy: od osobistych, przez komputerowe stanowiska robocze (ang. *workstations*), komputery średniej mocy obliczeniowe (jak np. Vax firmy DEC), duże komputery (ang. *mainframes*) i składające się z nich sieci.

Pakiety CASE są dość drogimi narzędziami programowymi: przy ich zakupie należy dokładnie przeanalizować potrzeby i cele stosowania narzędzia oraz dokładnie zbadać opcje oferowane przez dany pakiet. Należy sprawdzić, czy wyczerpuje on znane potrzeby, a także jakie daje dodatkowe możliwości. Do takiej analizy można zastosować odpowiednie formularze, takie jak zaproponowany w artykule [26] lub podany na rys. 1.

CASE W ZARZĄDZANIU

Dla potrzeb zarządzania w przemyśle i w handlu istnieje bardzo bogate oprogramowanie narzędziowe typu CASE, używane do wspomagania prac nad rozwojem oprogramowania. Oczywiście oprogramowanie tego rodzaju jest używane również i w innych dziedzinach, jednak w rozwiązaniach tego typu jest ono najbardziej rozbudowane, złożone i dzieli się na największą liczbę warstw. W artykule [15] zaproponowano podział na trzy poziomy: niski, średni i wysoki (ang. *upper, middle, lower* CASE): podział arbitralny, ale nie pozbawiony podstaw w rzeczywistości. Poszczególne poziomy mają następujące przeznaczenie:

- **wysoki,** nazywany też komputerowo wspomaganym planowaniem (ang. *Computer Aided Planning* – CAP), pozwala na przechowywanie w pamięci komputera i analizowanie informacji na temat struktury przedsiębiorstwa, jego celów, warunków osiągania tych celów itd.,
- **średni,** służący analizie i projektowaniu systemów informatycznych, w szczególności określeniu ich najbardziej efektywnej i odpowiadającej rzeczywistości struktury,
- **niski,** ukierunkowany na wspomaganie tworzenia programów realizujących funkcje opisane na wyższym poziomie w sposób co najmniej półautomatyczny, tzn. eliminujący konieczność pisania przez programistę fragmentów kodu realizujących funkcje powtarzalne w wielu systemach.

Poniżej tego ostatniego wyróżnia się jeszcze poziom zwany pod CASE, czyli całe coraz bogatsze instrumentarium programowe, wspomagające pracę analityka, projektanta i programisty.

Formularz oceny narzędzia CASE

INFORMACJA OGÓLNA
- Dostawca ☐
- Nazwa pakietu ☐
- Rok wprowadzenia ☐
- Rozpowszechnienie ☐
- Cena ☐

KOMPUTER DO IMPLEMENTACJI
- Firma produkująca ☐

KATEGORIA
- Pakiet narzędziowy ☐
- Pakiet zintegrowany ☐

TYP NARZĘDZI
- Planowanie ☐
- Analiza ☐
- Projektowanie ☐
- Projektowanie bazy danych ☐
- Projektowanie oprogramowania czasu rzeczywistego ☐
- Generator kodu ☐
- Programowanie ☐
- Utrzymanie oprogramowania ☐
- Otoczenie programowe ☐
- Zarządzanie pracami projektowymi ☐

GRAFIKA
- Kolor ☐
- Mysz ☐
- Okna ☐

REPOZYTORIUM PAKIETU CASE
- W centralnym komputerze ☐
- W komputerze osobistym ☐
- Architektura systemu sterowania bazą danych ☐
- Sprawozdawczość ☐
- Sterowanie zmianami ☐
- Śledzenie przebiegu ☐
- Sterowanie zmianami wersji ☐
- Zabezpieczenie stanu aktualnego ☐
- Logiczne rozdzielenie danych ☐
- Konsolidacja ☐

OPROGRAMOWANIE PROTOTYPOWANIA
- Projektowanie ekranów interakcji ☐
- Projektowanie raportów ☐
- Model funkcjonalny ☐
- Symulacja działania ☐

GENEROWANIE KODU
- Program szkieletowy ☐
- Pełny program ☐
- Język ☐
- Program interakcyjny ☐
- Program wsadowy ☐

OPROGRAMOWANIE PRZEBUDOWY SYSTEMÓW
- Analizator statyczny ☐
- Dokumentacja odnawiana ☐
- Modyfikacja struktury ☐
- Odtwarzanie ☐
- Analizator dynamiczny ☐
- Konwerter ☐

KONTROLA BŁĘDÓW
- Składnia ☐
- Zwartość ☐
- Kompletność ☐
- Zgodność z wymogami ☐
- Kontrola jakości ☐

OPROGRAMOWANIE PROCESU ROZWOJOWEGO PAKIETU
- Planowanie ☐
- Analiza ☐
- Projektowanie ☐
- Generowanie kodu ☐
- Utrzymanie w ruchu ☐
- Zarządzanie wytwarzaniem pakietu ☐

SYSTEM DOCELOWY
- Interakcyjny ☐
- Wsadowy ☐
- Przetwarzanie transakcji ☐
- Czasu rzeczywistego ☐
- Do wbudowania ☐

DOSTĘPNE SCHEMATY
- Przepływu danych ☐
- Sterowania ☐
- Tablice decyzyjne ☐
- Hierarchiczne struktury drzewiaste ☐
- Schemat strukturalny ☐
- Działania ☐
- Metoda Warniera-Orra ☐
- Przejścia między stanami ☐
- Pseudokod ☐
- Generator makiet ekranowych ☐
- Generator struktury interakcji ☐
- Generator sprawozdań ☐
- Struktura danych ☐
- Bytowo-relacyjny ☐
- Rekordy logiczne ☐
- Booch ☐
- Sieci Petriego ☐
- Inne ☐

PODSTAWY METODOLOGICZNE
- Yourdon ☐
- DeMarco ☐
- Gane/Sarson ☐
- Bachman ☐
- Chen ☐
- Martin ☐
- Merise ☐
- Orr ☐
- Jackson ☐
- Ward /Mellor ☐
- Hatley ☐
- Object-Oriented ☐
- SADT ☐
- Stradis ☐
- Method-1 ☐
- LSDM ☐
- Inne ☐

Rys. 1.

Jak widać całe programowanie, czyli to co przeciętny programista zwykł uważać za swoje główne zajęcie, stanowi zaledwie najniższy poziom, w dodatku nie wiadomo na pewno, czy w ogóle wymagający udziału człowieka.

Poziom wysoki

Osoby odpowiedzialne za zarządzanie różnymi przedsiębiorstwami spędzają wiele czasu na próbach zrozumienia wszystkich mechanizmów działania i określania planu przedsięwzięć firmy. Plany działania opisują cele, strategie ich osiągania, metody ogniskowania wysiłków na realizacji tych strategii oraz określają zasady oceny efektów realizacji zamierzeń. Do tych założeń dobiera się strukturę administracyjną, wyznacza się terminy realizacyjne i następstwo realizacji w czasie. Wyodrębnia się czynniki determinujące powodzenie, konieczne rozmieszczenie zasobów. Przewiduje się wpływ czynników zewnętrznych i pojawiające się w związku z tym problemy dla przedsiębiorstwa. Opracowuje się scenariusze postępowania, rozpatrując sytuacje najgorsze i najlepsze.

Wysoki poziom CASE dostarcza oprogramowania umożliwiającego opis przedsiębiorstwa i jego planów, na ogół z wykorzystaniem diagramów lub innych graficznych metod opisu. Przy użyciu tych metod dokonuje się dekompozycji ogólnego opisu przedsiębiorstwa na coraz bardziej szczegółowy opis jego składowych. Jednym z podstawowych elementów jest opis poszczególnych komórek organizacyjnych, ich funkcji i sposobów działania. Innymi elementami są: cele, odpowiedzialność, zasoby, otoczenie całego przedsiębiorstwa i jego składowych. Oprócz informacji graficznej, związanej ze strukturą analizowanego obiektu, użytkownik oprogramowania musi mieć dostęp do informacji tekstowej, opisującej na ogół właściwości elementów tej struktury.

Tak stworzone opisy mogą wspomagać tworzenie strategicznych planów działania. Istniejący opis na ogół umożliwia prezentację zasobów oraz zadań przedsiębiorstwa. Narzędzie z poziomu wysokiego CASE dostarcza opisu struktur, które użytkownik uszczegóławia, podając atrybuty, takie jak cele, ograniczenia, zasoby itd. Dla różnych planów struktura opisu przedsiębiorstwa pozostaje w zasadzie nie zmieniona, zmieniają się natomiast parametry.

Tworzenie takich modeli wymaga sporego nakładu urzędniczej pracy, jednak w miarę korzystania z systemu okazuje się, że raz wprowadzony opis może być wielokrotnie i w różnych celach wykorzystany. Bardzo ważne w systemach wysokiego poziomu CASE jest rozwiązanie sposobu komunikacji z użytkownikiem, zwłaszcza elastyczny i pozwalający na wielokrotne wykorzystanie elementów dostęp do słownika danych. Niektóre z systemów zawierają fragment umożliwiający wykorzystanie znanych od dość dawna metod planowania przedsięwzięć, takich jak metoda ścieżki krytycznej lub schematy Gantta.

Mogłoby się wydawać, że systemy z wysokiego poziomu CASE są przeznaczone głównie dla personelu kierowniczego przedsiębiorstwa, w którym je wdrożono, i że nie są zbytnio przydatne dla osób odpowiedzialnych za tworzenie i utrzymanie systemów informatycznych. Nic bardziej błędnego. Filozofia CASE obejmuje cały cykl życia oprogramowania, niezależnie od tego, jak nazwiemy poszczególne etapy tego cyklu. W szczególności CASE pozwala uzyskać wsparcie komputera w najważniejszej fazie tego cyklu: specyfikacji wymagań, projektu struktury, założeń technicznych, czy jak jeszcze można nazwać fazę zapoznawania się projektanta z obiektem. Decyzje podjęte w tej fazie stają się dość prędko nieodwracalne lub odwracalne bardzo dużym kosztem. Niezrozumienie wszystkich zasad działania obiektu musi prowadzić do stworzenia nieprawidłowego systemu informatycznego. Wysoki poziom CASE umożliwia nie tylko dostęp do pełnego i kompetentnego opisu obiektu, lecz również symulację zmian niezbędnych do przeprowadzenia przed informatyzacją części lub całej firmy. W dotychczasowej praktyce takich zmian na ogół się nie przeprowadzało, ponieważ zadaniem projektanta było dopasowanie systemu do zastanej rzeczywistości, a nie zmiana tej rzeczywistości.

Poziom średni

Średni poziom CASE wspomaga analizę problemów związanych z przetwarzaniem danych i projektowanie rozwiązań tych problemów. W większości oprogramowania tego poziomu korzysta się z graficznych metod projektowania i słownika danych podobnego do tych, które występują na wysokim poziomie. W odróżnieniu od narzędzi wysokiego poziomu, programy średniego poziomu wspierają pracę analityka systemów.

Oprogramowanie tego poziomu istotnie skraca czas rozwoju projektu oprogramowania użytkowego, przy czym wiedza, którą w przypadku stosowania narzędzi tradycyjnych mieli tylko członkowie zespołu projektującego system, jest w sposób pełny i systematyczny zapisana w odpowiedniej bazie danych.

Niestety, tylko nieznaczna część informacji zapisanej w systemie wysokiego poziomu CASE jest użyteczna na poziomie średnim. O ile system wysokiego poziomu CASE zawiera informacje na temat strategii, o tyle w strukturach średniego poziomu można przechowywać odpowiedzi na następujące pytania:

- jakie operacje są wykonywane w poszczególnych działach i dlaczego są to operacje ważne?
- dlaczego operacje wykonuje się w ten, a nie inny sposób?
- jakie informacje są potrzebne do wykonania tych operacji i w jaki sposób się je wykorzystuje?
- dlaczego określone warunki mają wpływ na wykonanie operacji oraz jakie informacje o tych warunkach są potrzebne i w jakim celu?
- kto odpowiada za poszczególne informacje?

Z drugiej strony tylko około 25% wiedzy zapamiętanej w bazach średniego poziomu CASE jest bezpośrednio przydatne na poziomie niskim, na którym dopiero tworzy się faktyczne specyfikacje programów. Narzędzia średniego poziomu pozwalają określić strukturę systemu informatycznego. Wobec niewielkiego powiązania z sąsiednimi warstwami neguje się niekiedy istnienie średniego poziomu CASE, rozdzielając jego funkcje na narzędzia poziomów niskiego i wysokiego. Należy jednak pamiętać, że o ile CASE wysokiego poziomu służy do opisu obiektów rzeczywistych, o tyle CASE średniego poziomu opisuje odpowiadające tym obiektom abstrakcje – struktury systemów informatycznych.

Poziom niski

Powrót od abstrakcji do konkretu następuje na niskim poziomie CASE, którego narzędzia służą do wspomagania projektowania i tworzenia programów komputerowych. Narzędzia tego poziomu są oparte, najogólniej rzecz biorąc, na jednym z dwóch podejść. W pierwszym z nich zakłada się, że podstawą tworzonej aplikacji jest jej struktura oraz funkcje. W drugim za pierwotne uważa się dane.

Osobiście uważam podejście od strony danych za bardziej naturalne i prostsze koncepcyjnie. Pierwszą czynnością przy projektowaniu jakiegokolwiek programu jest przy takim podejściu określenie zestawu danych wejściowych, wyjściowych, a także – w trakcie projektowania – danych pośrednich programu. Algorytmy działania programu są określane na podstawie wiedzy o obiektach, na których działają. Duża część fachowej wiedzy projektanta systemu informatycznego składa się z informacji o sposobach i możliwościach wprowadzania, przechowywania oraz przetwarzania danych o obiektach, na których działa program. Również projektant systemu informatycznego podejmuje na podstawie analizy (a czasem intuicyjnie) decyzje o sposobie agregowania obiektów różnych typów (na przykład – pól rekordów w rekord) lub tego samego typu (rekordów w plik bazy danych). Często decyzje podjęte w tym momencie mają zasadniczy wpływ na parametry eksploatacyjne finalnego produktu.

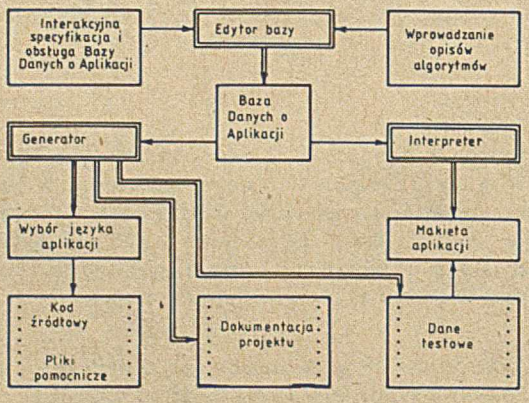
Narzędzia niskiego poziomu CASE pozwalają budować programy użytkowe (aplikacje) przy użyciu środków graficznych, na ogół z wyko-

rzystaniem słownika opisu aplikacji. Kolejność czynności wykonawcy (programisty?) przy tworzeniu takich aplikacji jest następująca:

1. Stworzenie słownika danych. Słownik danych zawiera opis wszystkich danych, struktur i dokumentów, z których będzie korzystać program użytkowy. Opis danej składa się między innymi z określenia: nazwy, typów (według różnych kryteriów: dana wejściowa-wewnętrzna-wyjściowa, tekstowa-numeryczna, jedno-wielokrotna itd.), maksymalnej liczby znaków potrzebnych do wprowadzenia lub wyświetlenia danej, wartości dopuszczalnych (dziedziny) itd.
2. Określenie formatów, według których dane będą wprowadzane i wyświetlane (formatki ekranowe) i drukowane (raporty).
3. Opisanie w sposób formalny algorytmów przetwarzania danych.
4. Określenie struktury aplikacji (systemu menu).
5. Stworzenie makiety systemu, którą można przedstawić odbiorcy – na przykład – do ćwiczeń.
6. Tworzenie dokumentacji projektu i lektura tejże wraz z uwagami z testowania makiety przez użytkownika.
7. Powtarzanie operacji 1–6 w trybie aktualizacji wcześniej wprowadzanych danych do czasu uzyskania zadowalającej makiety.
8. Wygenerowanie kodu programu użytkowego w języku docelowym oraz plików pomocniczych (na przykład dla programu typu MAKE, czy innych narzędzi klasy pod CASE).

Takie podejście pozwala połączyć zalety metody makietowania z korzyściami wynikającymi ze stosowania specjalizowanych języków bardzo wysokiego poziomu (ang. *fourth generation language*). W wyniku jego stosowania uzyskuje się automatycznie:
- makietę aplikacji,
- znaczny procent kodu źródłowego ostatecznej postaci aplikacji,
- dokumentację konstrukcyjną,
- powstałe w wyniku analizy powiązań między danymi oraz układu procedur pliki dla programów systemowych, umożliwiające wygenerowanie optymalnej, według pewnych kryteriów, postaci wykonywalnej aplikacji.

Przykładową strukturę narzędzi programowych do realizacji takich celów przedstawiono na rys. 2.



Rys. 2.

Centralnym punktem tego rysunku jest Baza Danych o Aplikacji. Opis aplikacji można przedstawić jako drzewo, którego gałęziami są kolejne operacje wykonywane przez program, a liśćmi dane wykorzystywane na tych poziomach. Ponieważ te same dane mogą być użyte w różnych gałęziach, w drzewie pojawiają się powiązania czyniące z niego sieć. Dane do Bazy Danych o Aplikacji są wprowadzane przy użyciu Edytora Bazy, przez osobę pracującą nad aplikacją.

Dla opisu struktur danych i funkcji projektowanej aplikacji stosuje się narzędzia programistyczne, oparte na poniżej opisanych wybranych technikach i metodach.

Metody

Przemysłowa produkcja oprogramowania wymaga stosowania metod i technik projektowania i implementowania systemów przez duże zespoły programistów. Techniki takie, rozwijane równolegle z tworzeniem produktów programowych, są sukcesywnie włączane do narzędzi CASE. Poniżej przedstawiono kilka wybranych metod, nazywanych czasem dumnie metodologiami. Problemem wykraczającym poza zakres tego artykułu, ale równie istotnym, są normy dotyczące dokumentowania wszystkich faz pracy nad produktem programowym [18]. Dopiero wspólne stosowanie opisanych technik i systematyczne dokumentowanie wszystkich etapów pracy, daje cień szansy na powodzenie realizacji naprawdę dużych projektów.

Metodologie w inżynierii programowania powstają niejako przy okazji rozwiązywania problemów konkretnych. Analitycy i programiści rozwiązują problemy, obserwują co robią inne grupy i w trakcie pracy, przechodząc od konkretu do abstrakcji, tworzą swoje metody pracy. Kiedy okazuje się (a ze względu na sposób działania ludzkiego mózgu musi się okazać), że kilka grup stosuje podobne metody, powstaje metodologia zwykle o nazwie pochodzącej od nazwiska osoby, która opublikowała opis metody [5, 11, 20, 22, 23, 24, 25, 27, 33]. Inni informatycy czytając taką publikację utwierdzają się w przekonaniu o słuszności swoich metod i, odprężeni, działają jeszcze efektywniej. Również programiści, którzy w ogóle nie czytają literatury fachowej, stosują nieświadomie techniki projektowania wstępującego lub zstępującego (albo wariant pośredni), bo inaczej po prostu się nie da. Praktyka dowodzi, że programista na ogół wie, że program działa, zanim (o ile w ogóle) uświadomi sobie dlaczego.

Z metodologiami są związane techniki prezentowania informacji. Z uwagi na największą przepustowość wzroku jako kanału informacyjnego u człowieka, większość technik opiera się na różnego rodzaju diagramach. Inną powszechnie stosowaną metodą jest rysowanie schematów działania programu, struktur danych, przepływu sterowania, tablic przejść stanów itd. Wszystkie te techniki połączone ze sobą stanowią podstawę działania narzędzi CASE.

METODA WARNIERA-ORRA

Opublikowany w 1972 roku w IBM System Journal artykuł Terry Bakera *"Chief Programmer Team Operation"* był pierwszą próbą połączenia kilku stosowanych w praktyce technik i metod: programowania strukturalnego, projektowania zstępującego, zespołu głównego programisty oraz bibliotekarza projektu. Artykuł ten zapoczątkował powszechne stosowanie technik strukturalnego projektowania i tworzenia aplikacji w USA.

W tym czasie Ken Orr, który również stosował techniki strukturalnego projektowania, doszedł do wniosku, że hierarchiczna struktura programów odzwierciedla w dużej mierze hierarchię danych tego programu, będących z kolei odpowiednikiem struktury opisywanej rzeczywistości. Równocześnie Jean Warnier przedstawił nie tylko podobne podejście [36], ale również zbudował systematyczną metodologię programowania opartego na strukturach danych. Orr w swoich pracach korzystał również z wyników Michaela Jacksona [19]. Syntezą tych prac stała się technika nazwana DSSD *(Data-Structured Systems Development* – tworzenie systemów na podstawie struktury danych), która stanowi podstawę metodologii znanej jako podejście Warniera-Orra [16, 28].

W podejściu Warniera-Orra przyjmuje się struktury danych jako szkielet do budowy programów oraz systemów. Programy są budowane hierarchicznie, z użyciem ograniczonej liczby prostych struktur danych. Taka metoda pozwala tworzyć poprawnie działające programy dla całych klas zastosowań, w których struktury danych wejściowych i wyjściowych są identyczne lub podobne. W celu rozwiązywania złożonych, wielopoziomowych problemów dekomponuje się je w taki sposób, żeby zawsze mieć do czynienia co najwyżej z jednopoziomową relacją *n:n* (wiele do wielu). Jest to podejście zgodne z tym, co robią od lat matematycy, rozbijając złożone problemy na elementarne, dla których istnieją jasne, jednoznaczne rozwiązania.

W przypadku projektowania opartego na strukturach danych kolejno następuje transformacja fizycznej struktury danych wejściowych w logiczną strukturę, logiczną strukturę danych wejściowych w logiczną

strukturę danych wyjściowych, i wreszcie logicznej struktury danych wyjściowych w ich strukturę fizyczną. Samo projektowanie odbywa się w tej metodzie w kierunku odwrotnym, to znaczy najpierw określa się fizyczny format danych wyjściowych (na ogół wiadomo, czego żąda się od systemu), aby przez określenie logicznej struktury danych wewnętrznych i ich transformacji dojść do wymagań dotyczących fizycznej struktury wejścia.

Przy wykonywaniu bardziej złożonych systemów okazuje się dość szybko, że główne problemy leżą nie w samym programowaniu, ale na poziomie projektowania systemu. Pojawia się pytanie: jak projektować zbiór programów działających na wspólnych danych? Stosowanie techniki SDDS jest pomocne w próbach odpowiedzi na takie pytanie. Jej stosowanie wspomaga projektowanie struktur bazy danych, określanie wymogów dotyczących systemu, określanie architektury systemu. Na tym poziomie opisywania technika zachowuje swoją podstawową cechę: „w tył" od wyjścia. Różnica polega na tym, że z fizycznych struktur danych wyjściowych przechodzi się nie do idealnych struktur danych, ale do określenia bazy danych, wspólnej dla całego systemu. Taka baza ma w tej metodzie postać znormalizowanej bazy relacyjnej.

Określenie rezultatów działania programu przez zdefiniowanie danych wyjściowych oraz algorytmów stanowi dobrą podstawę do projektowania systemu, ale nie jest wystarczające do sformułowania wymagań. W miarę rozwoju metody tworzenia systemów na podstawie struktur danych przyjęto określanie najpierw kontekstu, potem funkcji, a na końcu rezultatów działania systemu. Techniki wspomagające te działania to: schematy bytów (ang. *entity diagram)*, diagramy Warniera-Orra (ang. *assembly-line diagrams)*, pozwalające określić zależność między funkcjami systemu, logiczne projektowanie danych i inne.

Przykładem narzędzia korzystającego z tej metodologii jest Design Machine, opracowana w firmie Kena Orra – Optima Inc., Schaumburg, IL, USA.

METODA GANE-SARSONA

Zaproponowane przez Gane'a i Sarsona podejście do projektowania systemów bywa nazywane modelowaniem logicznym [14]. Jego celem jest szybkie uzyskanie precyzyjnych definicji elementów systemu z niejasnych i nieprecyzyjnych początkowo informacji o przyszłych danych i funkcjach projektowanego systemu. Istotny wpływ na efektywność metody ma zastosowanie technik graficznych, umożliwiających przedstawienie struktury danych i sterowania w systemie, bez uciekania się do fizycznej implementacji lub makietowania systemu.

W pracy [13] modelowanie logiczne przedstawiono jako proces składający się z siedmiu kroków. Pierwszym krokiem jest opracowanie diagramów przepływu danych (ang. *Data-Flow Diagram* – DFD). Schemat przepływu danych jest grafem, którego węzłami są:
- byty (ang. *entities*), np. klienci, dostawcy, działy przedsiębiorstwa,
- składnice danych (w końcowym systemie pliki danych),
- procesy (czynności do wykonywania danych).

Krawędziami grafu łączącymi byty z procesami i składnicami danych są linie ze strzałkami określającymi kierunek przesyłania informacji. Linie są zawsze jednostronnie skierowane. Jeśli jest przekazywana informacja zwrotna, to używa się kolejnych linii. Informacja przepływu od bytów do procesów, od procesów do składnic informacji, od składnic do procesów i od procesów do bytów.

Przedstawiona technika ma trzy istotne zalety:
- pozwala ściśle określić zakres systemu – byty zewnętrzne (klienci, banki, dostawcy) są z definicji poza zakresem systemu, procesy nie przedstawione na diagramie nie są częścią projektu,
- diagram jest czytelny dla osób nie zajmujących się informatyką, a więc dla większości zleceniodawców,
- w sposób przejrzysty ukazuje zarówno dane systemu, jak i procesy przekształcające te dane, unaoczniając zależności między danymi i procesami.

Drugim krokiem jest zestawienie listy elementów danych wraz z informacją, w którym magazynie danych dana powinna być przechowywana. Staranne wykonanie tego kroku pozwala uniknąć redundancji danych i zagregować dane powiązane ze sobą w jednym miejscu.

Trzecim krokiem jest przeprowadzenie analizy relacji między danymi i uszczegółowienie struktur logicznych baz danych.

W czwartym kroku dane przedstawia się w postaci połączonych, dwuwymiarowych tablic i określa się klucze, według których powinien odbywać się dostęp do wierszy tych tablic. Podobnie jak w poprzedniej metodzie wykorzystuje się tu relacyjny model danych.

Piątym krokiem jest aktualizacja schematu przepływu danych na podstawie znormalizowanych relacji, uzyskanych w kroku czwartym. Ten krok jest faktycznie ostatnim krokiem modelowania logicznego. Dwa pozostałe służą już projektowaniu systemu informatycznego, działającego na wyspecyfikowanych danych.

Szóstym krokiem jest wydzielenie procesów i związanych z nimi danych, tak aby można było określić jednostki proceduralne projektowanego systemu. W tym etapie należy odpowiedzieć na pytania:
- kiedy jest wykonywana procedura?
- jaki obszar diagramu przepływu danych jest zaangażowany w jej realizacji?
- czy ten obszar może być zrealizowany przy użyciu jednej procedury, a jeśli nie to dlaczego?

Siódmym krokiem jest fizyczne projektowanie samych procedur, wspierane sformalizowanymi informacjami zebranymi w krokach poprzednich.

Współtwórca metody, Chris Gane, jest konsultantem firmy Bachman Information Systems, Cambridge, MA, USA.

PODEJŚCIE RELACYJNO-BYTOWE

Jednym z głównych czynników opóźniających wdrażanie zintegrowanych systemów informatycznych jest to, że na ogół systemy takie są projektowane i wykonywane stopniowo, a nie w sposób globalny dla danego przedsiębiorstwa. Stosowanie zintegrowanej bazy danych, wokół której buduje się kolejne podsystemy, jest jakimś rozwiązaniem, z tym że sam system bazy danych nie zapewnia jeszcze poprawnego porządkowania danych. Metoda relacji bytów (ang. *entity-relationship approach*) jest techniką pozwalającą w sposób systematyczny przetwarzać wymagania użytkownika w schematy prawidłowo zaprojektowanych baz danych [6, 9, 21, 31, 32].

Dostęp do elementów danych, zapisanych w bazie odbywa się na ogół przez nazwę. Dość często przy tworzeniu systemu informatycznego pojawia się potrzeba rozbudowy systemu na podstawie już istniejących baz danych. W sytuacji tworzenia przez więcej niż jeden zespół projektantów systemów działających na podobnych danych pojawia się niebezpieczeństwo wystąpienia następujących problemów:
- różnych nazw dla tego samego fizycznie elementu danych,
- tych samych nazw dla różnych elementów danych,
- niezgodności formatów danych,
- redundancji danych,
- nieprawidłowości aktualizacji.

Równocześnie zawsze występuje problem liczby typów rekordów (w przypadku modelu relacyjnego – relacji), jakie powinny wystąpić w bazie. Nie zawsze jest oczywiste, które pole lub kombinacja pól winny być kluczem. Metoda relacji bytów (stosowana nieformalnie i nieświadomie przez praktycznie każdego projektującego struktury danych) jest obecnie najpopularniejszą techniką projektowania takich relacji.

Graficzną techniką wspomagającą projektowanie jest w tej metodzie Diagram relacji bytów (ang. *Entity-Relationship Diagram* – ERD). Byty w tym diagramie reprezentują obiekty, organizacje, zdarzenia i inne elementy opisu rzeczywistości, które mają być przechowywane w bazie systemu. Byty są połączone liniami zaopatrzonymi w strzałki. W tym schemacie strzałki oznaczają relacje między bytami. Relacje te mogą być dwustronne. Musi istnieć graficzny wyróżnik, informujący czy relacja jest jedno- czy wielokrotna.

Analiza diagramu relacji bytów pozwala opisać zawartość bazy w kategoriach relacyjnego modelu baz danych. Przykładowo, strukturę danych przedsiębiorstwa tworzącego programy można opisać następująco:

PRACOWNIA (ID_PRACOWNI, OBŁOŻENIE)
PROGRAMISTA (ID_PROGRAMISTY, NAZWISKO, WIEK, ID_PRACOWNI)

PROJEKT (NR_PROJEKTU, NAZWA_PROJEKTU)
PRACUJE_W (ID_PROGRAMISTY, NR_PROJEKTU, %CZASU)

Słowa przed nawiasami odpowiadają albo bytom (PRACOWNIA, PROGRAMISTA, PROJEKT) albo relacjom (PRACUJE_W). Słowa w nawiasach są atrybutami, pozwalającymi tworzyć klucze, umożliwiające dostęp do danych. Z punktu widzenia teorii relacyjnych baz danych byty są tu również relacjami. Wszystkie relacje występują w trzeciej postaci normalnej.

Trzecim krokiem jest opracowanie aplikacji, z wykorzystaniem narzędzi zbliżonych do języka zapytań systemowych (ang. *System Query Language* – SQL). SQL jest narzędziem związanym z relacyjnymi bazami danych, metodę ER natomiast można stosować dla różnych modeli bazy danych. Jeśli mamy do czynienia z innym modelem, to można wykorzystywać metody zapytań właściwe dla danego modelu. Istnieje norma ANSI [2], dotycząca słowników zasobów informacyjnych, oparta na relacyjnym modelu bazy danych i diagramach relacji bytów.

METODA YOURDONA

Metoda Yourdona jest efektem dwudziestoletnich doświadczeń wielu ludzi współpracujących z Edwardem Yourdonem, między innymi w firmie Yourdon Inc. Techniki składające się na tę metodę są nazywane: strukturalną analizą, strukturalnym projektowaniem i strukturalnym programowaniem. Metodologia Yourdona stale ewoluuje pod wpływem idei innych informatyków, poczynając od modnych w latach siedemdziesiątych technik strukturalnych, przez ciągle rozwijane techniki diagramów, po elementy podejścia obiektowego [17, 29, 35, 37, 38].

Metodologia Yourdona składa się z dwóch części: narzędzi i technik. Narzędziami są różnego rodzaju diagramy graficzne, używane do przyrostowego modelowania wymagań i architektury projektowanych systemów.

Najpopularniejszym narzędziem jest diagram przepływu danych (DFD), opisany przy okazji omawiania metody Gane-Sarsona. O ile diagram przepływu danych jest doskonałym narzędziem opisu funkcji systemu, o tyle niewiele mówi o zależnościach między danymi i o uwarunkowaniach czasowych systemu. W celu uwzględnienia tych czynników stosuje się w metodzie Yourdona diagramy relacji bytów (ERD) oraz diagramy zmian stanów (ang. state-transition diagram – STD). Różnego rodzaju diagramy dostarczają czytelnej i bogatej wiedzy o projektowanym systemie, jednak wiedza ta nie jest pełna. Kompletny opis wymaga również informacji tekstowej, między innymi o strukturach elementów danych oraz działaniu procesów, opisywanych w DFD pojedynczym bloczkiem.

O ile opisane narzędzia ułatwiają istotnie analizę systemu, o tyle techniki zaproponowane przez Yourdona mają ułatwić proces tworzenia aplikacji. Podstawową techniką jest rozdział zdarzeń (ang. *event partitioning*). Polega ona na stworzeniu diagramu kontekstu (ang. *context diagram*) wysokiego poziomu. Diagram taki służy do określenia kontekstu systemu, jego granic oraz wzajemnych oddziaływań ze światem zewnętrznym. Następnie tworzy się wykaz zdarzeń, które zachodzą w otoczeniu systemu, a na które system powinien reagować. Praktycznie takie zdarzenia są na ogół transakcjami bazy danych systemu. Na podstawie wykazu opracowuje się diagram przepływu danych. Z kolei diagram ten redukuje się, łącząc na przykład zdarzenia, które powodują korzystanie z takich samych danych. Redukcja diagramu prowadzi w końcu do określenia fizycznej struktury aplikacji.

METODY TWORZENIA SYSTEMÓW CZASU RZECZYWISTEGO

W przypadku stosowania technik CASE w systemach czasu rzeczywistego pojawiają się problemy o mniejszym na ogół statycznym stopniu złożoności, ale bardzo skomplikowane w związku z wystąpieniem czasu jako podstawowego parametru w systemie. Istnieje kilka sposobów podejścia do tworzenia systemów czasu rzeczywistego [12].

Pierwszy z nich każe traktować system jako jeden wielki proces, ulegający w trakcie dekompozycji na coraz mniejsze podprocesy. Takie podejście określa się mianem funkcjonalnej dekompozycji. Z drugiej strony znane podprocesy podstawowe mogą być kompletowane w większe. Wynikowy model może być przedstawiony jako hierarchia diagramów przepływu danych, na przykład w rodzaju zaproponowanych przez DeMarco [10]. Takie podejście od strony hierarchi funkcji, zostało rozwinięte dla systemów czasu rzeczywistego między innymi przez Hatleya. Pierwotne dla określenia struktury systemu informatycznego są procesy, potem zależności między nimi, a w końcu dane. U podstaw tej metody leżą dobrze opracowane techniki projektowania i programowania strukturalnego.

Diametralnie różne podejście zaproponowali w 1984 Ward i Mellor [35]. System jest przez nich widziany jako „czarna skrzynka", która dostarcza ustalonych odpowiedzi na zewnętrzne sygnały. Projektant systemu definiuje zdarzenia zachodzące na zewnątrz systemu (zewnętrzne pobudzenia) i określa odpowiedzi, definiując oddzielne procesy, odpowiedzialne za wywołanie tych reakcji. W tym podejściu pierwotne są zdarzenia zewnętrzne, procesy są efektem zajścia tych zdarzeń, a dane procesów są całkowicie pochodne w stosunku do funkcji procesu. Zaletą podejścia Ward-Mellora jest szybkie przejście osoby realizującej system do określania jego działań. Kiedy już istnieje model, dość prostą rzeczą jest jego rozbudowa o nowe zdarzenia lub zmiana reakcji na już zdefiniowane zdarzenia. Podstawową wadą jest duża podatność innych procesów na zmiany w jednym – wprowadzenie nowego procesu może nieść konieczność zmian w danych i działaniach innych.

W trzecim wreszcie podejściu, stosowanym w produktach firmy Project Technology, z Berkeley, świat jest traktowany jako nieograniczony i wypełniony obiektami. Reguły określające zależności między obiektami są ściśle określone. Każdy obiekt i każda relacja, w której obiekt bierze udział, mają określony czas życia. Formalizacją cykli życia obiektów są tablice przejść stanów, dokładnie opisujące stany, w których mogą znajdować się obiekty, oraz zdarzenia, powodujące zmianę stanu obiektu. Definiowanie aplikacji zaczyna się od określenia danych, w oderwaniu od ich fizycznego znaczenia. Z kolei definiuje się sterowanie przez odwzorowanie fizycznych właściwości obiektu. Procesy są definiowane automatycznie na podstawie dwóch poprzednich kroków. Takie podejście pozwala na proste rozszerzenie liczby obiektów lub procesów, bez zaburzeń w innych częściach systemu. W systemie projektowanym nie jest sprawą prostą i oczywistą precyzyjne określenie co dzieje się, kiedy system jest poddany ekstremalnym i złożonym obciążeniom. W odróżnieniu od podejścia poprzedniego, zachowanie „czarnych skrzynek" nie jest opisane w jednym miejscu projektowanego systemu.

Innym narzędziem, coraz częściej wykorzystywanym przy projektowaniu systemów czasu rzeczywistego, jest SDL (ang. *Specification and Description Language*), będący standardowym językiem specyfikacji i opisu systemów. Został opracowany i poddany standaryzacji przez CCITT (*International Telegraph and Telephone Consultative Comitee*). Rozpoczęte w 1972 roku prace zakończyły się, jak na razie, przyjęciem w 1987 roku wersji SDL88. Początkowo SDL stosowano do opisu systemów telekomunikacyjnych, obecnie znajduje on coraz powszechniejsze zastosowanie we wspomaganiu dowolnych systemów czasu rzeczywistego. Przykładem narzędzia opartego na języku SDL, wykorzystującego wszystkie możliwości współczesnych systemów CASE, jest program SDT szwedzkiej firmy TeleLOGIC.

MAKIETOWANIE

Makietowanie (ang. *rapid prototyping*) [1, 3, 4, 7, 8, 34] nie jest oddzielną metodologią, a raczej sposobem dość szybkiego uzyskiwania działającego modelu aplikacji. W odróżnieniu od proponowanego – na przykład – przez Gane'a modelowania działania aplikacji, makietowanie jest związane z generowaniem lub z interpretacją opisu elementów aplikacji, szczególnie dotyczących interakcji programu z użytkownikiem. Programy stosujące technikę makietowania pozwalają modyfikować opisy elementów interakcji (menu, formatki, podpowiedzi) i natychmiast przedstawić efekt tego działania na ekranie. Taka technika pozwala na udział końcowego użytkownika w procesie tworzenia aplikacji, bez konieczności przygotowywania na papierze projektów ekranów, raportów, menu. Technikę makietowania stosuje się w wielu narzędziach korzystających opisanych metodologii.

W pewnym sensie narzędzia używane do makietowania są podobne do wspomnianych już języków czwartej generacji, w których oprócz normalnych elementów języka programowania są wbudowane silne

narzędzia dostępu do informacji w bazach danych. Podobnie narzędzia wspomagające są makietowane swego rodzaju interpreterami języków opisu interakcji z użytkownikiem. Po uzyskaniu właściwej postaci aplikacji jest generowany pseudokod lub wręcz kod programu w języku programowania, który następnie może być, w różnym stopniu – zależnym od możliwości narzędzia makietującego – uzupełniany ręcznie o fragmenty dotyczące przetwarzania, które to fragmenty z kolei mogą pochodzić z innych narzędzi generowania kodu.

LITERATURA

[1] Alavi M.: An Assessment of the Prototyping Approach to Information Systems Development. Communications of the ACM, Vol. 27, No. 6, pp. 556–563, 1984
[2] ANSI. Standards on Information Resource Dictionary Systems, 1988
[3] Appleton D. S.: Data-Driven Prototyping. Datamation, pp. 259–268, 1983
[4] Boar B. H.: Application Prototyping – A Requirements Definition Strategy for the 80 s. John Wiley, New York 1984
[5] Boehm B.: Improving Software Productivity. Computer, No. 8, 1987
[6] Chen P. P.: Entity-Relationship Model – Towards a Unified View of Data. ACM Trans. on Database Systems, Vol. 1, No. 1
[7] Connel J., Brice L.: Rapid Prototyping. Datamation, Vol. 30, No. 4, 1984
[8] Connel J., Schafer L.: Rapid Prototyping. Yourdon Press /Prentice-Hall, New York 1989
[9] Davies C. G. S., Jajodia P. Ng., Yeh. R.: Entity-Relationship Approach to Software Engineering. North-Holland, New York 1983
[10] DeMarco T.: Structured Analysis and Systems Specification. Prentice-Hall, Englewood Cliffs (NJ) 1979
[11] Dolin K.: Business Computer Systems Design. Mitchell Publ., Santa Cruz (CA) 1984
[12] Falk H.: CASE Tools Emerge to Handle Real-time Systems. Computer Design. January 1, pp. 53–74, 1988
[13] Gane C.: Rapid System Development. Prentice-Hall, Englewood Cliffs (NJ), 1988
[14] Gane C., Sarson T.: Structured system analysis – Tools and Techniques. Prentice-Hall, Englewood Cliffs (NJ), 1979
[15] Gibson M. L.: The CASE Philosophy. BYTE, No. 4, pp. 209–218, 1989
[16] Hansen K.: Data Structured Program Design. 2nd ed., Prentice-Hall, Englewood Cliffs (NJ), 1986
[17] Higgins A. A.: Designing Structured Programs. Prentice-Hall, Englewood Cliffs, 1983
[18] IEEE Guide to Software Requirements Specifications. ANSI/IEEE Std. 830–1984, Institute of Electrical and Electronical Engeneers Inc., 1984
[19] Jackson M. A.: Principles of Program Design. Academic Press. New York, 1975
[20] King D.: Current Practices in Software Development. Yourdon Press, Englewood Cliffs (NJ), 1984
[21] March S. (Ed.): Entity Relationship Approach. North Holland, New York, 1988
[22] Martin C., F.: User Centered Requirements Analysis Prentice-Hall, Englewood Cliffs (NJ), 1988
[23] Martin J., McClure C.: Diagramming Techniques for Analysts and Programmers. Prentice-Hall, Englewood Cliffs (NJ), 1985
[24] Martin J., McClure C.: Action Diagrams: Clearly Structured Specifications, Programs and Procedures. 2nd ed. Prentice-Hall, Englewood Cliffs (NJ) 1989
[25] Mączyński J.: Papers on Program Design. Prace IPPT, Nr 5, Warszawa, 1983
[26] McClure C.: The CASE Experience. BYTE, pp. 235–246, April 1989
[27] Orr K. T., Gane Ch., Yourdon E., Chen P. P., Constantine L. L.: Methodology: The Experts Speaks. BYTE, pp. 221–233, April 1989
[28] Orr K. T.: Structured Requirements Definition. Topeka, KS, KEN Orr&Ass., 1980
[29] Page-Jones M.: The Practical Guide to Structured Systems Design. 2nd ed. Yourdon Press /Prentice-Hall, New York 1988
[30] Rook P.: Controlling software projects. Software Engineering Journal, No. 1, 1986
[31] Smith M. C.: Database Design: Composing Fully Normalized Tables from a Rigorous Dependency Diagram: Communications of the ACM, Vol. 28, No. 8, 1985
[32] Teorey T., Fry J.: An extended Entity-Relationship Approach to Logical Database Design. ACM Survey, 1986
[33] Wallace S.: Methodology: CASE's Critical Cornestone. Bussiness Software Review, April 1988
[34] Wang Yu: A Distributed Specification Model and Its Prototyping. IEEE Trans. on Software Engineering, No. 8, 1988
[35] Ward P., Mellor St.: Structured Techniques for Real-Time Systems. Yourdon Press /Prentice Hall, New York 1985
[36] Warnier J. D.: Logical Construction of Systems. Van Nostrand Reinhold, New York, 1981
[37] Yourdon E.: Modern Structured Analysis. Yourdon Press /Prentice-Hall, New York, 1989
[38] Yourdon E., Constantine L. L.: Structured Design. Yourdon Press /Prentice Hall, Englewood Cliffs (NJ), 1987.

Informujemy Czytelników! Cena normalna INFORMATYKI w roku 1991 pozostaje bez zmiany – 5800 zł, cena ulgowa została podwyższona do 2900 zł za egzemplarz.

Nowe sterowniki sieci

Rozwój sieci komputerowych dopinguje producentów półprzewodników do opracowywania nowych układów sterujących siecią Ethernet o coraz większym stopniu inteligencji i scalenia. Nowe układy tego typu oferowały już firmy Advanced Micro. Devices Inc. oraz Intel Corp., a ostatnio dołączyła do nich również firma National Semiconductor Corp. z propozycją układu o nazwie SONIC (*System-Oriented Network-Interface Controller* – ukierunkowany na systemy sterownik sprzęgu sieci). Rozwiązuje on wiele problemów występujących w dotychczasowych sterownikach. Polegały one na tym, że komputer centralny był w znacznym stopniu obciążony funkcjami protokołu sieci; ponadto sterowniki te nie uwzględniały takich funkcji zarządzania siecią, jak ograniczanie dostępu do węzłów lub określenie liczby kolizji. Funkcje te stają się bardzo istotne ze wzrostem liczby węzłów. Inną wadą dotychczasowych sterowników było zmniejszenie wydajności całego systemu wskutek blokowania magistrali systemowej oraz żądania przy przesyłaniu danych interwencji procesora centralnego. Dołączenia do sieci wymagały dotychczas dodatkowej karty, natomiast SONIC zapewnia inteligentne i tanie sterowanie siecią Ethernet w karcie głównej komputera osobistego. Układ ten służy także jako tani i bardzo wydajny łącznik między różnymi sieciami. Maksymalna prędkość przesyłania wynosi 80 MB/s, a więc 25-krotnie więcej niż w większości obecnie używanych sterowników. Dzięki tak dużej prędkości przesyłania, wydajność systemu nie zmniejsza się, nawet w przypadku dołączenia do magistrali dodatkowych urządzeń peryferyjnych.

SONIC jest pierwszym sterownikiem sieci lokalnych, który w jednej kostce zawiera analogowo-cyfrowy koder-dekoder oraz realizuje funkcje cyfrowe. Pozwala to zbudować na dwu kostkach (SONIC + układ dekodujący, zrealizowany na programowalnej matrycy), pełny sterownik sieci Ethernet, podczas gdy poprzednio urządzenie takie, konstruowane na karcie adapterowej, wymagało aż 10–15 kostek.

Ze względu na dominację w komputerach osobistych dwóch rodzin mikroprocesorów: Intela 80X86 oraz kontroli 680X0, SONIC został dostosowany do parametrów obu tych rodzin. Ma on dodatkową końcówkę oraz układy logiczne, zapewniające właściwy sprzęg z zastosowanym procesorem. Rozwiązanie takie powoduje, że operacje SONIC-a ulegają zmianom, dostosowując się do określonej architektury magistrali. Oprócz tego urządzenie może współpracować zarówno z mikroprocesorami 16-, jak i 32-bitowymi obydwu firm.

Niektóre funkcje zarządzania siecią są zrealizowane sprzętowo. Sterowanie złączem logicznym pozwala na określenie w danym węźle punktów wielokrotnego dostępu. Zapewnia to sterowanie dostępem do węzłów oraz ułatwia zarządzającym siecią realizację takich funkcji, jak np. ograniczenia dostępu dla różnych użytkowników. Możliwe jest też gromadzenie informacji o pracy węzła, sygnalizacja zwarć kabla itp. awarii, a także danych statystycznych dotyczących kolizji błędów, co może być wykorzystane do podejmowania prawidłowych decyzji o podziale sieci. Układ usprawnia również działanie mostów sieciowych, eliminując konieczność buforowania tych projektów, które nie są przesyłane między sieciami.

(JR)

Pierwsza wersja tego artykułu została przedstawiona podczas Trzeciej Wiosennej Szkoły PTI pn. „Komputerowe wspomaganie tworzenia systemów informatycznych", która odbyła się w dniach 14–18 maja 1990 r. w Świnoujściu.

KRZYSZTOF DWORAKOWSKI, HENRYK KLIMEK, TADEUSZ KORNIAK, ELŻBIETA PRZEPIÓRA
Przedsiębiorstwo Techniczno-Handlowe BISTER
Katowice

Przechowywanie i przetwarzanie danych o złożonych nieregularnych strukturach

W artykule opisano podstawowe cechy oprogramowania systemu zarządzania hierarchiczną bazą danych GRBASE, przeznaczonego do obsługi danych o nieregularnych i nie dających się z góry określić strukturach. Przedstawiono zarys struktury danych bazy GRBASE, możliwości funkcjonalne systemu zarządzania, a także informacje o zrealizowanej przykładowej aplikacji.

W tradycyjnych bazach danych nie uwzględnia się możliwości łatwego tworzenia dynamicznie zmieniających się struktur danych. Obecnie są podejmowane próby zaadaptowania istniejących baz danych do zarządzania tego rodzaju obiektami, wiąże się to jednak z małą efektywnością, ze względu na specyficzne właściwości przechowywanych danych. Odpowiadając na istniejące zapotrzebowanie, system GRBASE zapewnia efektywność zarządzania danymi o dynamicznych strukturach drzewiastych.

Baza danych GRBASE służy do przechowywania i przetwarzania danych o nieregularnej strukturze i powiązaniach hierarchicznych. System zarządzania bazą danych GRBASE umożliwia obsługę danych użytkownika w zakresie tworzenia, modyfikowania, wyszukiwania i przetwarzania obiektów Bazy. Przewidziano również kontrolę dostępu do obiektów.

Oprogramowanie jest przeznaczone dla komputerów zgodnych z IBM PC/XT oraz IBM PC/AT z systemem operacyjnym MS-DOS (dla wersji wielodostępnej również z systemem Xenix).

Wykorzystując oprogramowanie SZBD GRBASE, użytkownik (programista aplikacyjny) może łączyć dane w określone przez siebie grupy (segmentacja danych). Między tak utworzonymi grupami danych (segmentami) można dowolnie ustalać hierarchiczną strukturę drzewiastą. Każdy zespół segmentów, połączonych w drzewo, stanowi obiekt bazy danych.

Korzystając z oferowanych środków programowych programista aplikacyjny może realizować, między innymi, następujące operacje:

- definiowanie i modyfikowanie obiektów, zarówno ich struktury, jak i wchodzących w ich skład segmentów;
- definiowanie zestawów atrybutów dla obiektów oraz ich modyfikowanie (dopisywanie, usuwanie, aktualizowanie);
- wyszukiwanie obiektów lub ich grup, spełniających określone warunki.

Oprogramowanie GRBASE stanowi więc zestaw narzędzi umożliwiających tworzenie systemów aplikacyjnych, w których istotną rolę odgrywają złożone i zmieniające się w dużym stopniu powiązania między danymi.

Dane i ich struktury

Baza danych GRBASE służy do przechowywania i przetwarzania obiektów o strukturze drzewa. Należy rozumieć, iż informacja użytkownika jest skupiona w węzłach drzewa, a powiązania między węzłami ustalają hierarchię między danymi. Obiekty bazy danych GRBASE mają opisaną poniżej specyficzną budowę.

Dane elementarne

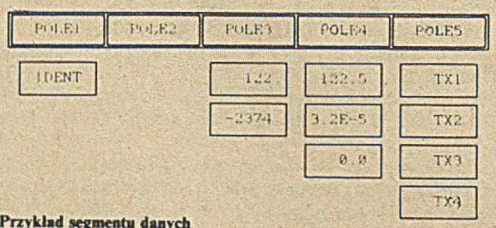
Jest to najmniejsza jednostka danych, udostępniana użytkownikowi środkami systemowymi. W systemie zaimplementowano między innymi dane typu INTEGER, REAL, TEXT, DATE, CHAR.

Pole danych

Polem danych nazywa się dowolną sekwencję (być może o krotności 0) danych elementarnych tego samego typu. Poszczególne dane elementarne wchodzące w skład pola są nazywane jego wystąpieniami.

Segment

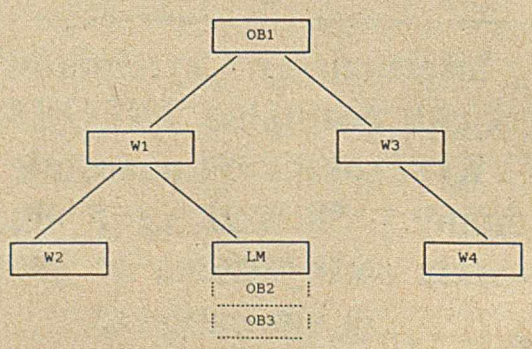
Przez segment (segment danych) rozumie się sekwencję nazwanych pól. Zakłada się, że nazwy pól w ramach segmentu będą niepowtarzalne. Dwa segmenty mają ten sam format, jeśli składają się z sekwencji pól o tych samych nazwach i typach. Segment stanowi podstawową jednostkę danych użytkownika. Dostęp do poszczególnych wystąpień pól w segmencie użytkownik uzyskuje przez zadanie nazwy pola (lub jego numeru) i numeru wystąpienia pola. Szczególnym przypadkiem segmentu jest tzw. **prymityw**, będący segmentem danych o wyróżnionym formacie. Postać segmentu danych zilustrowano na rys. 1.



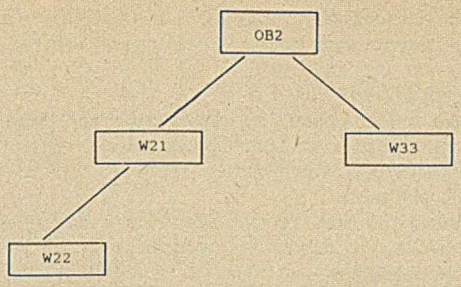
Rys. 1. Przykład segmentu danych

Obiekt Bazy Danych

Formalnie obiekt projektowanej bazy danych należy rozumieć jako drzewo z rozróżnialnymi przez nazwy węzłami. Należy w tym miejscu podkreślić, że poszczególne obiekty bazy, traktowane jako drzewa, nie muszą być izomorficzne (w sensie izoformizmu grafów). Przykłady obiektów bazy danych podano na rysunkach 2, 3 i 4 – występujące tam pojęcie makroprymitywu jest opisane w dalszej części tekstu.



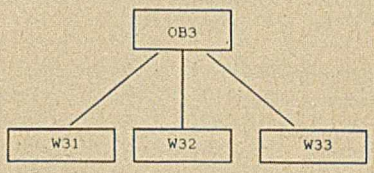
Rys. 2. Obiekt z liściem makroprymitywów



Rys. 3. Obiekt bazy danych

Z logicznego (użytkowego) punktu widzenia wśród węzłów obiektu wyróżnia się następujące:

- korzeń,
- węzeł prymitywów,
- liść makroprymitywów,
- gałązka.



Rys. 4. Obiekt bazy danych

Korzeń

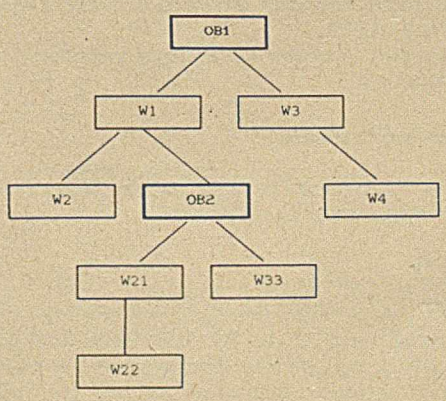
Jest rozumiany w sensie korzenia grafu. Informacją użytkową skupioną w korzeniu jest pojedynczy segment danych. Korzeniom różnych obiektów odpowiadają różne segmenty danych, przy czym wszystkie te segmenty mają ten sam format. Korzenie (a tym samym obiekty Bazy Danych) są rozróżnialne – niepowtarzalnym identyfikatorem jest zawartość pierwszego pola segmentu danych korzenia.

Węzeł prymitywów

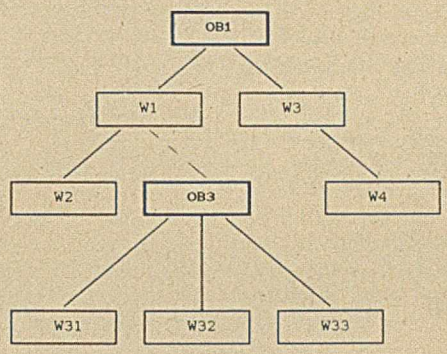
Węzeł prymitywów jest węzłem obiektu, z którym jest związana lista (kolekcja) prymitywów. Poszczególne prymitywy kolekcji mogą znajdować się w różnych plikach prymitywów. Ten sam prymityw może występować w różnych kolekcjach obiektu, jak również w kolekcjach różnych obiektów. W danej chwili tylko jeden prymityw z kolekcji (wybrany przez użytkownika) może być dostępny do przetwarzania.

Liść makroprymitywów

Liść makroprymitywów jest węzłem obiektu o szczególnym znaczeniu i nie jest z nim bezpośrednio związana informacja użytkowa (segment lub prymityw danych). Z liściem makroprymitywów jest związana lista obiektów (kolekcja makroprymitywów). Użytkownik wskazuje, który z makroprymitywów kolekcji jest w danej chwili widoczny (dołączony do obiektu). W obiektach wykorzystywanych jako makroprymitywy mogą również występować liście makroprymitywów, co pozwala na wielopiętrową konstrukcję obiektów bazy. Liść makroprymitywów jest zawsze węzłem wiszącym obiektu. Na rysunku 5 zilustrowano obiekt OB1 z rys. 2 w sytuacji, gdy widocznym makroprymitywem jest obiekt OB2 z rys. 3. Na rysunku 6 pokazano ten sam obiekt z widocznym makroprymitywem OB3 z rys. 4.



Rys. 5. Obiekt OB1 z widocznym makroprymitywem OB2



Rys. 6. Obiekt OB1 z widocznym makroprymitywem OB3

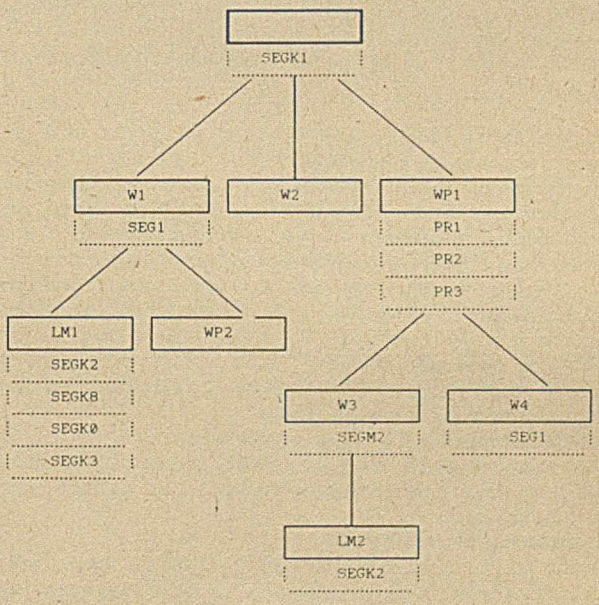
Gałązka

Gałązką nazywa się węzeł obiektu nie będący żadnym z uprzednio omówionych. Informacją użytkową skupioną w gałązce jest pojedynczy segment danych. Różnym gałązkom w tym samym obiekcie, jak również gałązkom różnych obiektów, mogą odpowiadać te same segmenty danych.

Różnica między poszczególnymi rodzajami węzłów w grafie polega na opisanym wyżej ich usytuowaniu w strukturze drzewa oraz trybie dostępu do informacji użytkowej związanej z danym węzłem. Udostępnienie węzła obiektu (udostępnienie informacji związanej z węzłem) jest rozumiane w zależności od jego rodzaju:

- dla korzenia i gałązki dostępny staje się segment danych przypisany węzłowi,
- dla węzła prymitywów i liścia makroprymitywów dostępnymi stają się operacje umożliwiające przejście do wskazanego elementu kolekcji i operacje jego przetwarzania (w tym sensie można powiedzieć, że – w odróżnieniu od korzenia i gałązki – informacja użytkowa nie jest bezpośrednio skupiona w liściu).

Przykład rozbudowanego obiektu, z różnego rodzaju węzłami, przedstawiono na rys. 7.



Rys. 7. Przykład rozbudowanego obiektu bazy danych

W przykładzie obiektu z rys. 7 węzły (nazwy węzłów w pogrubionych ramkach) mają następujące znaczenie.

- **Węzeł oznaczony pustą ramką** jest korzeniem obiektu z przypisanym segmentem danych SEGK1, ustalonego jednolicie dla wszystkich

obiektów bazy formatu. Nazwa SEGK1 stanowi identyfikator obiektu. Pusta ramka na rysunku odzwierciedla fakt, iż korzeniowi każdego obiektu system przypisuje nazwę złożoną ze znaków NULL.

- **Węzeł W1** jest gałązką z przypisanym segmentem SEG1 przechowywanym w pliku segmentów.
- **Węzeł LM1** jest liściem makroprymitywów. Kolekcję tworzą obiekty o identyfikatorach SEGK2, SEGK8, SEGK0 i SEGK3.
- **Węzeł WP2** jest węzłem prymitywów. Aktualnie kolekcja prymitywów jest pusta.
- **Węzeł W2** jest „pustą" gałązką, tzn. gałązką, do której nie jest przypisany żaden segment danych.
- **Węzeł WP1** jest węzłem prymitywów. W kolekcji znajdują się prymitywy PR1, PR2 i PR3.
- **Węzeł W3** jest gałązką z dołączonym segmentem SEGM2 przechowywanym w pliku danych.
- **Węzeł LM2** jest liściem makroprymitywów. Kolekcja składa się z pojedynczego elementu (obiekt o identyfikatorze SEGK2).
- **Węzeł W4** jest gałązką z dołączonym segmentem SEG1.

Warto zauważyć, że segment danych SEG1 został dołączony do dwóch węzłów W1 i W4. Nie oznacza to oczywiście jego fizycznego zdublowania w bazie.

Przewiduje się, że część informacji związanej z obiektem będzie miała charakter dynamiczny, tzn. będzie zależna nie tylko od samego obiektu, lecz również od sytuacji, w jakiej ten obiekt jest udostępniany użytkownikowi. Na informację, o której mowa powyżej, składa się zestaw danych elementarnych (parametrów), z których każda może być powtarzalna i które są ładowane do ustalonych pól korzenia w chwili gdy obiekt staje się widoczny jako makroprymityw. Każdemu wystąpieniu obiektu w kolekcji makroprymitywów jest przypisany indywidualny zestaw parametrów. Poszczególne zestawy parametrów mogą się oczywiście różnić między sobą.

Parametry pozwalają na zróżnicowane widzenie tego samego obiektu, co – mając zwłaszcza na uwadze konieczność pogodzenia potrzeb różnych użytkowników – uelastycznia możliwości przetwarzania bazy danych.

Usługi realizowane przez GRBASE

System spełnia, za pośrednictwem oferowanych usług, następujące podstawowe funkcje obsługi bazy:

- tworzenie bazy danych,
- tworzenie obiektów bazy,
- organizowanie dostępu do obiektów bazy i ich części składowych,
- rejestrowanie wykonywanych operacji,
- organizowanie przeszukiwania i selekcjonowania zbioru obiektów bazy.

System oferuje następujące grupy usług (operacji):

Operacje organizujące bazę danych
Do operacji tych należy m.in. tworzenie bazy, definiowanie formatów segmentów oraz rejestrowanie użytkowników. Operacje tej grupy są realizowane przez samodzielne programy usługowe.

Operacje organizacyjne procesu przetwarzania
Są to głównie operacje rozpoczynające lub kończące przetwarzanie bazy. Operacje tej i pozostałych grup są realizowane przez procedury Biblioteki Procedur GRBASE.

Operacje na obiektach i węzłach
Operacje tej grupy umożliwiają tworzenie, modyfikowanie struktury oraz usuwanie obiektów bazy danych.

Operacje dotyczące liści i węzłów prymitywów
Jest to specjalna grupa operacji, umożliwiająca dostęp do elementów kolekcji związanej z liściem makroprymitywów lub węzłem prymitywów i ich przetwarzanie.

Porządkowanie zbioru obiektów
Operacje tej grupy ustalają obowiązujący porządek (przez wskazanie aktywnego klucza) w zbiorze obiektów, co umożliwia zorganizowane poruszanie się po tym zbiorze.

Wyszukiwanie obiektów
Są to operacje udostępniające użytkownikowi obiekt spełniający warunki zadane w postaci związków między polami korzenia lub segmentów węzła.

Przesłanianie obiektów
Operacje te pozwalają na logiczne wydzielenie z całej bazy podzbioru obiektów spełniających zadany przez użytkownika warunek. Pozostałe obiekty stają się wtedy „niewidoczne" i nie biorą udziału w przetwarzaniu.

Poruszanie się po obiekcie
Są to operacje umożliwiające poruszanie się (we wskazany przez użytkownika sposób) po węzłach dostępnego obiektu. Węzeł docelowy zostaje udostępniony użytkownikowi.

Operacje na danych węzła
Ta grupa operacji pozwala czytać, zapisywać i modyfikować pola segmentu związanego z dostępnym węzłem. Jest również możliwa wymiana całego segmentu.

Operacje informacyjne
Funkcje tej grupy podają charakterystyki liczbowe związane ze zbiorem obiektów lub pojedynczym obiektem oraz realizują wybrane operacje arytmetyczne na polach wskazanych segmentów.

Zestaw operacji dobrano tak, aby umożliwić użytkownikowi sprawną obsługę i przetwarzanie hierarchicznej bazy danych.

Niezależnie od podstawowego trybu wykorzystywania usług (operacji) SZBD GRBASE, za jaki należy uważać stosowanie w programach aplikacyjnych procedur z Biblioteki Procedur GRBASE, jest również możliwy konwersacyjny tryb pracy z bazą.

Program Interpretatora Zleceń GRBASE umożliwia użytkownikowi interakcyjne wywoływanie poszczególnych operacji (w sposób zbliżony do stosowanego w dBase III Plus). Pozwoli to użytkownikowi nie będącemu programistą wykonywać praktycznie wszystkie operacje z Biblioteki Procedur. Przy tym trybie wykorzystania realizowanego oprogramowania jest stosowany zestaw środków wspomagających pracę użytkownika (kontekstowe objaśnienia, pełnoekranowa wizualizacja danych itp.).

Ponadto, w rozszerzonej wersji systemu GRBASE zrealizowano dwa mechanizmy:

- wielodostępu,
- ochrony danych.

Mechanizm wielodostępu umożliwia równoległe przetwarzanie tej samej bazy danych przez wielu użytkowników. System zapewnia bezpieczny dostęp do obiektów bazy i danych (segmentów, prymitywów), unikając jednocześnie nadmiernego blokowania tych elementów.

Mechanizm ochrony danych pozwala traktować selektywnie uprawnienia różnych użytkowników do obiektów bazy danych. Dla każdego z użytkowników bazy jest określony podzbiór jej obiektów, do którego ten użytkownik ma dostęp.

Przykładowa aplikacja systemu GRBASE

Analizując dokładnie strukturę danych opisujących rzeczywiste obiekty, można się przekonać, że w gruncie rzeczy układają się one w nieregularne, drzewiaste struktury. Aby to zbadać, stworzono bardzo uproszczony model systemu informatyzacji przedsiębiorstwa – hipotetycznego Wydziału Klocków Fabryki Zabawek. Wyróżniono typowe zakresy stosowania systemów informatycznych, wśród których dwa główne to:

- wspomagane projektowanie produktu złożonego z wielu elementów (przez analizę do TPP),
- analiza wyników finansowych (analogia systemów tworzonych dla działu planowania).

Wspomaganie projektowania

Produktem Wydziału są Zestawy Klocków. Każdy Zestaw składa się ze ściśle określonej liczby klocków.

Model Główny

Podstawową cechą Zestawu jest to, że można z niego zbudować ściśle określony model, tzw. Model Główny, charakterystyczny dla tego Zestawu. Rysunek tego modelu widnieje na opakowaniu i w katalogu załączanym do Zestawu. Dopuszcza się, aby po złożeniu Modelu Głównego została pewna, niewielka liczba klocków luzem.

Zestawienie materiałowe

Ostatnia strona katalogu zawiera rysunek wszystkich klocków wchodzących w skład Zestawu (jest także podana ich liczba). Jest to więc swoiste zestawienie materiałowe Zestawu.

Sposób projektowania nowego Zestawu

Projekt nowego Zestawu utworzony przez Projektanta obejmuje między innymi:
- zestawienie materiałowe, tj. liczbę i rodzaj wszystkich klocków wchodzących w skład Zestawu (Model Główny może wymagać mniej klocków, niż podano w Zestawie, aby można z niego było budować atrakcyjne inne modele),
- wstępna kalkulacja kosztów Zestawu oparta na średnich cenach klocków wchodzących w jego skład.

Analiza wyników finansowych

Danymi wejściowymi do tej części systemu są faktury wystawiane klientom na zamawiane partie zestawów. Dla uproszczenia pominięto cały dział planowania produkcji na podstawie zamówień, jest on analogiczny do części rozliczeniowej. Głównym celem tej części jest:
- wystawianie Klientom faktur na podstawie informacji o wysłanych partiach dostarczanych przez Dział Kompletowania i Magazynów,
- analiza opłacalności sprzedaży poszczególnych Zestawów,
- zmiana kalkulacji cen poszczególnych Zestawów pod wpływem zmiany ceny jednego z jego elementów.

Dane opisowe zawarto w segmentach o strukturze przedstawionej na rys. 8.

Faktura (Sprzedaż):

KOD_SPRZ	DATA_SPRZ	KOD_ZEST	LICZ_ZEST	CENA_UMOWN	ZYSK
C10	D8	C10	N5	N6.2	N6.2

Zestaw:

KOD_ZEST	NAZWA_ZEST	DATA_ZATW	KOSZT_ZEST
C10	C20	D8	N6.2

Klocek:

KOD_KLOC	NAZWA_KLOC	WYMIARY_KL	KOSZT_KLOC	DATA_ZATW
C10	C20	N3	N6.2	D8

Pracownik:

KOD_PRAC	NAZWISKO	IMIE	DATA_URODZ	DATA_ZATRU	PREMIA
C10	C20	C10	D8	D8	N6.2

Klient:

KOD_KLIE	NAZWA_KLIE	MIASTO	ADRES
C10	C40	C40	C40

Parametry:

KLUCZ	LICZBA
C10	i*N4

Rys. 8. Struktura poszczególnych segmentów

Obiekty danych

Obiekty bazy odzwierciedlają strukturę typowych zapytań systemu. Identyfikatorem obiektu jest dziesięcioznakowy klucz rozpoczynający się dwuliterowym przedrostkiem 11_.

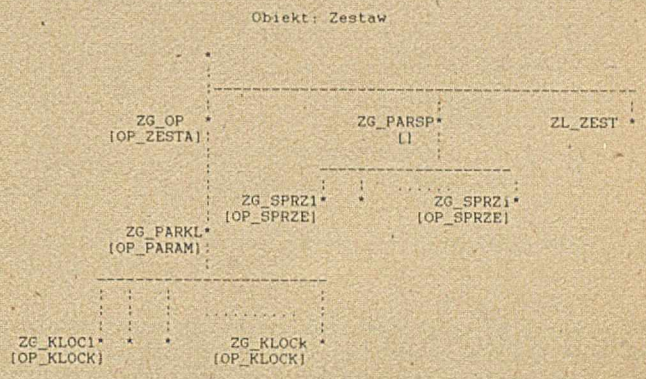
Pierwsza litera definiuje obiekt:
Z – Zestaw
E – Klocek
K – Klient
S – Faktura (Sprzedaż)
P – Pracownik

druga natomiast określa rodzaj węzła:
G – gałązka
L – liść makroprymitywów

Druga część nazwy jest zazwyczaj związana z typem segmentu związanego z węzłem (jeśli jest) lub funkcją węzła, np. ZG_KLOC1. Każdy z segmentów (z wyjątkiem segmentu parametrów, niejako sztucznym) rozpoczyna się polem KOD_... . Pole to zawiera powtórzony jednoznaczny klucz obiektu nadawany kolejno sekwencyjnie przez system.

Dzięki takiemu rozwiązaniu do obiektów-nie-właścicieli segmenty mogą być łatwo wstępnie uporządkowane wg narastających kodów. Pozwoli to na szybkie przeszukiwanie kolekcji liści lub węzłów-dzieci, bez tworzenia zbyt skomplikowanych algorytmów.

Konsekwencją tej metody jest, że kod raz zarejestrowany w systemie obiektu jest nieusuwalny, tzn. żaden inny obiekt go nie dostanie. Dziewięć znaków kodu wystarczy całkowicie do takich celów. Pierwsza litera kodu jest literą definiującą obiekt.

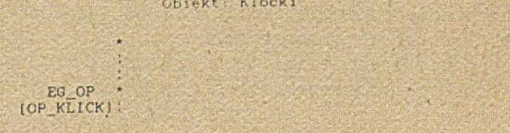


Rys. 9. Struktura obiektu Zestaw

Obiekt Zestaw składa się z trzech gałęzi (rys. 9):
1) gałęzi opisu technicznego, na którą składa się:
- węzeł-gałązka ZG_OP (segment OP_ZESTA) opisujący dane stałe zestawu,
- węzeł-gałązka ZG_PARKL (segment OP_PARAM) zastępujący nie zaimplementowane w tej wersji GRBASE parametry w ten sposób, że *i*-te wystąpienie pola LICZBA segmentu OP_PARAM określających liczbę klocków opisywanych przez *i*-te dziecko,
- *k* węzłów (liści) ZG_KLOCKI – ZG_KLOCKk z segmentami OP_KLOCK opisującymi klocki, z których składa się Zestaw;

2) gałęzi opisu sprzedaży, na którą składa się:
- węzeł-gałązka ZG_PARSP, nie związany z żadnym segmentem; służy on wydzieleniu części opisującej sprzedaż, dla urody oraz aby np. wykorzystać polecenie PODAJ_LICZBE_DZIECI; nie jest on potrzebny i jeśli będzie przeszkadzał lub opóźniał działanie, to można go usunąć,
- *i* węzłów (liści) ZG_SPRZ1-ZG_SPRZ*i* z segmentami OP_SPRZE opisującymi kolejne sprzedaże Zestaw;

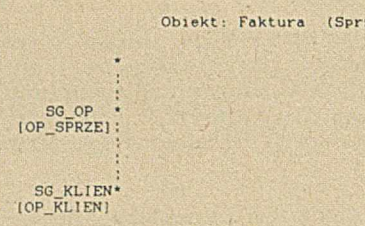
3) gałęzi zawierającej liść makroprymitywu ZL_TEST zawierającej kolekcję makroprymitywów-zestawów będących modelami wtórnymi.



Rys. 10. Struktura obiektu Klocki

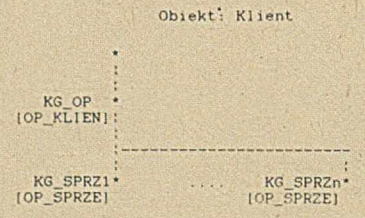
Obiekt Klocki (rys. 10) składa się z węzła-gałązki EG_OP (segment OP_SPRZE) i opisuje dane stałe klocka. Obiekt wprowadzono po to, aby w łatwy sposób dokonywać zmian w segmentach opisujących klocek.

Obiekt Faktura (rys. 11) składa się z węzłów:
- węzła-gałązki SG_OP (segment OP_SPRZE), opisującego dane stałe zestawu,
- węzła-liścia SG_KLIEN, opisującego Klienta, którego dotyczy sprzedaż.



Rys. 11. Struktura obiektu Faktura (Sprzedaż)

Obiekt Klient (rys. 12) składa się z węzłów:
- węzła-gałązki KG_OP (segment OP_SPRZE), opisującego dane stałe zestawu,
- węzeł-liści KG_SPRZ*i*, opisujący kolejne zakupy Klienta.



Rys. 12. Struktura obiektu Klient

Przykładowa aplikacja, aczkolwiek jest bardzo dużym uproszczeniem, pozwala domniemywać, że w rzeczywistych systemach przetwarzania wydają się występować struktury drzewiaste, bowiem rzeczywiste dane z dużym trudem mieszczą się w relizacyjnym modelu danych.

EUROGRAPHICS '91

W dniach 2–6 września 1991 r. na zamku Hofburg w Wiedniu odbędzie się kolejna, 12. międzynarodowa konferencja Europejskiego Towarzystwa Grafiki Komputerowej (European Association for Computer Graphics). Językiem oficjalnym konferencji będzie angielski. Tematyka konferencji:
- synteza i animacja obrazu *(Image Synthesis and Animation)*
- obrazowanie w nauce *(Scientific Visualization)*
- sprzężenie człowiek-komputer *(Human-Computer Interaction)*
- modelowanie *(Modelling)*
- sprzęt grafiki *(Graphics Hardware)*
- standardy grafiki *(Graphics Standards)*
- przetwarzanie obrazów *(Image Processing)*
- zagadnienia różne *(Miscellaneous)*.

Pierwsze dwa dni konferencji przeznaczono na seminaria prowadzone – na poziomie wstępnym i zaawansowanym – przez wybitnych specjalistów. Kolejne dni to prezentacja nadesłanych referatów na temat osiągnięć naukowych lub osiągnięć praktycznych, komunikatów oraz wyników programów grafiki finansowanych przez EWG. Międzynarodowe jury wyłoni najlepszy referat do nagrody Günthera Enderle w wysokości 1000 fr. szwajc. oraz kolejne trzy, przeznaczone do opublikowania w czasopiśmie Computers & Graphics.

W dniach 4–6 września odbędzie się wystawa grafiki komputerowej i jej zastosowań, a także konkurs generowanych komputerem taśm wideo, filmów i slajdów. Organizatorzy zapraszają do nadsyłania referatów i komunikatów w terminie do 10.12.90.

Szczegółowych informacji udziela: **Conference Secretariat, EUROGRAPHICS '91, c/o Interconvention, Austria Center Vienna, A–1450 VIENNA, tel. +43(1) 2369 2640, faks +43(1) 2369 648.**

Posiadasz maszyny

typu IBM, serii SM, PDP-11, MERA400...,

pracujące pod systemami operacyjnymi:

UNIX, XENIX, RSX, PC-MOS, MULTILINK, CONCURRENT PCDOS, QNX, CROOK...

To z całą pewnością potrzebujesz

TERMINALE ALFANUMERYCZNE

MT220

Skontaktuj się z nami.

MICRONET

Zakłady Elektroniczne
"MICRONET"
81-836 Sopot ul. Krasickiego 9
tel. 51-13-17 tlx 051-2876 mcrnt

0/17/90

ZAPRASZAMY

Konferencje

Computer Networks '91

W dniach 26–28 czerwca 1991 r. Ośrodek Obliczeniowy Politechniki Wrocławskiej organizuje czwartą cykliczną międzynarodową konferencję **COMPUTER NETWORKS**. Poprzednie trzy odbyły się w latach 1985, 1987 i 1989. Komitet programowy Konferencji określił następujące tematy referatów:
- metodologia projektowania systemów sieciowych,
- rozwój sprzętu sieciowego,
- połączenia sieciowe,
- standardy sieciowe,
- zastosowania sieci lokalnych,
- rozproszone bazy danych,
- zastosowania biurowe,
- interfejsy użytkowników sieci,
- systemy obsługi komunikatów,
- zarządzanie i nadzorowanie sieci,
- metody opisu formalnego,
- testowanie dostosowania,
- systemy baz wiedzy w sieciach komputerowych,
- ochrona, uwiarygodnianie i bezpieczeństwo danych,
- udział wielkich przedsiębiorstw w rozwoju sieci,
- eksploatacja i administrowanie,
- nowe usługi i technologie telekomunikacyjne,
- sieci szkół wyższych i placówek naukowych,
- sieci łączności przemysłowej,
- zarządzanie sieciami dla opieki zdrowotnej oraz klęsk żywiołowych,
- rozwój sieci w Europie Środkowej i Wschodniej.

Informacje oraz zgłaszanie referatów i uczestnictwa w Konferencji: **dr Andrzej Kalis, Politechnika Wrocławska, Wybrzeże Wyspiańskiego 27, 50–370 Wrocław, tel. 20-36-70, teleks 712254 polwr.**

MIECZYSŁAW KŁOPOTEK
Instytut Podstaw Informatyki PAN
Warszawa

Medytor
mówiący edytor języka polskiego

Wprowadzenie techniki komputerowej do wspomagania prac biurowych na całym świecie służyło nie tylko usprawnieniu procesu przygotowania tekstu, ale także poprawieniu ich estetycznej i gramatycznej jakości. W Polsce możliwości komputerów w tej dziedzinie są – niestety – wykorzystane w niewielkim stopniu. Stosowanie edytorów opracowanych dla innych języków niż polski powoduje, że polskie znaki są często niedostępne, brak jest koniecznej kontroli ortograficznej i automatycznego przenoszenia wyrazów zgodnie z regułami języka polskiego oraz innych udogodnień oferowanych przez tego rodzaju oprogramowanie.

Opracowany w Instytucie Podstaw Informatyki PAN edytor – o nazwie Medytor – ma na celu wyeliminowanie tych i wielu innych wad dostępnych edytorów.

Charakteryzuje się on:
- pełnym zestawem polskich znaków na ekranie monitora i na drukarce (z symulacją tych znaków w razie ich braku w posiadanym przez użytkownika sprzęcie),
- dynamicznym formatowaniem tekstu z uwzględnieniem polskich zasad przenoszenia wyrazów,
- możliwością kontroli ortograficznej (porównanie tekstu ze słownikiem) dla podzbioru języka polskiego,
- możliwością parametryzacji tekstu,
- możliwością warunkowego generowania wariantów pisma (zwroty zależne od płci, stanowiska itp.),
- możliwością zmiany szerokości oraz pozycji wiersza, na papierze i ekranie,
- automatycznym wcinaniem tekstu punktów i podpunktów,
- możliwością głosowego odczytania tekstu przez komputer (dla osób niewidomych lub słabo widzących).

Właściwości i zastosowanie programu Medytor

Medytor umożliwia przygotowanie tekstów sparametryzowanych i wielowariantowych, które następnie można wraz z parametrami generować za pomocą procedury vihelp ( ) w programie użytkowym lub za pomocą samodzielnego programu HELPDEMO (wtedy parametry są przekazywane przez COMMAND LINE). Edytor ten powstał w celu przygotowania wielowariantowych tekstów objaśnień dla opracowanego w IPI PAN systemu ekspertowo-statystycznego – SYS8688 [3].

W przygotowywanym tekście miejsce przywołania określonego parametru zaznacza się przez:

CTR_P ⟨identyfikator parametru⟩

Warunkowo przygotowany tekst ma następującą składnię:

CTR_W ⟨ident. param.⟩ [⟨operator porównania⟩] ⟨wartość porównywana⟩ ⟨generowany. tekst⟩ CTR–V

Przykład

Tekst podstawowy:

```
Informujemy
    CTR_Płeć;m*: Pana
        CTR_Tyt_nauk;prof.: Profesora CTR_V
        CTR_Tyt_nauk;doc.: Docenta CTR_V
        CTR_Tyt_nauk;~prof.: CTR_Tyt_nauk;~doc.:
                st_nauk;dr*: Doktorze CTR_V
        CTR_V CTR_V
    CTR_V
    CTR_Płeć;k*: Panią

        CTR_Tyt_nauk;prof.: Profesor CTR_V
        CTR_Tyt_nauk;doc.: Docent CTR_V
        CTR_Tyt_nauk;~prof.: CTR_Tyt_nauk;~doc.:
                st_nauk;dr*: Doktor CTR_V
        CTR_V CTR_V
    CTR_V
,że kopie artykułu pt.
CTR_Ptytuł; prześlemy na CTR_Płeć;m*:Pański CTR_V CTR_Płeć;k*:
Pani CTR_V adres.
```

Przy zadanych parametrach *płeć* =mężczyzna, *tyt_nauk* =prof., *tytuł* =„Tensory", otrzymuje się tekst:

Informujemy Pana Profesora, że kopie artykułu pt. „Tensory" prześlemy na Pański adres,

a dla parametrów *płeć* =kobieta, *tyt_nauk* =x (tj. brak tytułu), *st_nauk* =dr, *tytuł* =„Surfaktanty" – tekst:

Informujemy Panią Doktor, że kopie artykułu pt. „Surfaktanty" prześlemy na Pani adres.

Jak widać, warunek ma prostą postać: tekst podstawiony za parametr musi być leksykograficznie zgodny z tekstem wartości porównywanej. Jeżeli wartość porównywaną poprzedza operator ~ to wymagana jest niezgodność, a dla operatorów ⟨oraz⟩ zachowanie stosownego ścisłego porządku leksykograficznego. Znak ? jest zgodny z każdym znakiem, a znak * jest zgodny z resztą tekstu.

Operatory ⟩ oraz ⟨ mają zastosowanie także do parametrów numerycznych.

Przykład

Tekst podstawowy:

```
Korelacja wynosi CTR_Pkorelacja; i jest
    CTR_Wpoz_ist;>0.05;nieistotna CTR_V
    CTR_Wpoz_ist;~>0.05:
        CTR_Wpoz_ist;>0.01;stosunkowo istotna CTR_V
    CTR_V
    CTR_Wpoz_ist;~>0.01;
        CTR_Wpoz_ist;>0.001; istotna CTR_V
    CTR_V
    CTR_Wpoz_ist;~>0.001; bardzo istotna CTR_V
```

Po podstawieniu parametrów korelacja = 0.2 i poz_ist (tzn. poziom istotności) = 0.043, otrzymuje się:

Korelacja wynosi 0.2 *i jest stosunkowo istotna.*

Jest także możliwe odwrócenie ról: tekst podstawowy jest nośnikiem parametru, a sam parametr jest „sparametryzowany". W tekście podstawowym parametr jest generowany przez wyrażenie:

CTR_T ⟨nazwa parametru⟩ ⟨tekst podstawienia⟩,

a wprowadzoną definicję anuluje się wyrażeniem:

CTR_T ⟨nazwa parametru⟩

Ten efekt jest użyteczny wówczas, gdy tekst podstawowy ma sterować np. przypadkiem deklinacyjnym zadawanego parametru.

Przykład

Tekst podstawowy:

Wczoraj widziałem się z CTR_Tprzyp;Narzędnik; CTR_Posoba;. CTR_T|przyp;

Nie pozdrowiłem CTR_Tprzyp;Dopełniacz; CTR_Posoba; CTR_T|przyp;

Po podstawieniu parametru

```
os.    - JanCTR_Wprzyp;Dop*;aCTR_V
         CTR_Wprzyp;Cel*;owiCTR_V
         CTR_Wprzyp;Bier*;aCTR_V
         CTR_Wprzyp;Na*;emCTR_V
         CTR_Wprzyp;Nie*;eCTR_V
         CTR_Wprzyp;Wo*;eCTR_V
        NowakCTR_Wprzyp;Dop*;aCTR_V
         CTR_Wprzyp;Cel*;owiCTR_V
         CTR_Wprzyp;Bier*;aCTR_V
         CTR_Wprzyp;Na*;emCTR_V
         CTR_Wprzyp;Nie*;aCTR_V
```

otrzymuje się:

Wczoraj widziałem się z Janem Nowakiem.
Nie pozdrowiłem Jana Nowaka.

W wielu zastosowaniach trzeba wyróżnić fragment tekstu przez przedstawienie go jako kolumny o innej szerokości niż reszta tekstu (np. normalny tekst jest pisany w kolumnach 10–64, a wybrany dłuższy cytat z innej pracy – w kolumnach 25–54). Podobnie w oficjalnych pismach, w nagłówku własny adres wpisuje się np. w kolumnach 5–40, adresata – w kolumnach 40–70, a właściwy tekst – w kolumnach 15–65. Medytor umożliwia tego rodzaju rozmieszczanie tekstu przez umieszczenie w danym wierszu znaków CTR_O i CTR_Q, co określa granice „okna", w którym będzie pisany dalszy ciąg tekstu (do następnej pary znaków CTR_O:CTR_Q. Jest także możliwe warunkowe ustawianie szerokości okna, np. zależnie od szerokości stosowanego papieru (72 lub 80 kolumn).

Przy formowaniu tekstu są uwzględniane reguły języka polskiego. Formatowanie jest dynamiczne, tzn. następuje dopiero po podstawieniu parametrów i sprawdzeniu warunków generowania. W ten sposób nie powstają w tekście luki, wynikające z faktu pojawienia się np. zbyt krótkiego nazwiska, ani też obcięcia np. zbyt długiego imienia. Przyimki i inne słowa z zadanej listy nigdy nie są umieszczane na końcu wiersza. Formatowanie odbywa się do znaku wymuszonej zmiany wiersza (w Medytorze – CTR_A).

Jeżeli pierwszym znakiem nowego akapitu jest litera, to akapit ten zaczyna się wcięciem tekstu, a wiersze następne są pisane od brzegu. Jeżeli pierwszym znakiem jest natomiast cyfra lub znak – (a więc zaczyna się podpunkt w tekście), to następuje wcięcie. Przy formatowaniu nie powstaje efekt odsunięcia pierwszego słowa tytułu od cyfr oznaczających numer rozdziału, a następne wiersze tytułu zaczynają się dokładnie pod pierwszym słowem tytułu. Specjalny znak sterujący przewidziano dla tych akapitów, które pod żadnym pozorem nie mogą być formatowane (tabele, wzory itp.).

Medytor zapewnia kontrolę poprawności ortograficznej tekstu polegającą na porównaniu tekstu ze słownikiem. W razie stwierdzenia niezgodności można zażądać wyszukania słów najbardziej podobnych, zignorować ostrzeżenie, uzupełnić słownik lub poprawić dane słowo.

Możliwość odczytania tekstu na głos przez komputer ma niewątpliwie znaczenie dla osób słabo widzących. Odczytanie tekstu na głos jest również pewną namiastką kontroli syntaktycznej, gdyż niweluje efekt autosugestii ze strony osoby przygotowującej tekst.

Wybrane zagadnienia implementacji

Większość technik implementacji edytorów jest znana lub oczywista. Warto tu zwrócić jedynie uwagę na interesujące aspekty kontroli ortograficznej oraz otoczenia polskojęzycznego. Problematykę generowania mowy omówiono w [2]. W Medytorze odczytywanie na głos jest inicjowane przez ustawianie kursora na pierwszym słowie interesującego fragmentu i naciśnięcie klawisza funkcyjnego (F6). System czyta tekst do końca pliku lub do chwili przerwania przez naciśnięcie dowolnego klawisza.

System kontroli ortograficznej powinien obejmować dwa istotne elementy: odpowiednio obszerny słownik języka wraz z systemem dostępu (wyszukiwania) oraz właściwą strategię konstruowania słów najbardziej podobnych do analizowanego słowa, jeśli nie występuje ono w słowniku.

Analogiczne słowniki w popularnych systemach kontroli ortograficznej dla języków angielskiego [1] lub niemieckiego [4] obejmują około 100 000 – 200 000 słów. Duży słownik języka polskiego powinien docelowo obejmować co najmniej zbliżoną liczbę słów. Słowniki tego rodzaju rezydują na ogół w pamięci operacyjnej komputera. Przy średniej długości słów angielskich (ok. 6 znaków) słownik taki z trudem mieściłby się np. w pamięci IBM PC o pojemności 640 KB. Pamięć o takiej pojemności na pewno jednak nie pomieści 100 000 słów polskich, których średnia długość jest znacznie większa.

Dlatego ważna jest kompresja słownika. Zyskuje ona prymat nad metodami szybkiego dostępu do posortowanego słownika (metodą „dziel i rządź"), gdyż czas dostępu do dysku jest dużo dłuższy niż czas dostępu do pamięci operacyjnej. Jedną z możliwych technik jest wykorzystanie wspólnych początkowych części słów słownika. Zamiast całych słów przechowuje się tylko tę część słowa, która różni się od słowa poprzedniego w słowniku oraz długość części wspólnej.

Przykład

Słowo	Przechowywana część
kobiałka	kobiałka
kobiecość	ecość
kobieta	eta
kowal	wal

Gdyby słowa składały się z zupełnie dowolnych sekwencji liter, to dla słownika zawierającego 100 000 słów można by zaoszczędzić w ten sposób tylko 38 000 bajtów pamięci. Jednak liczba liter, które mogą nastąpić po danej literze w słowie jest ograniczona (do średnio nie więcej niż 10 liter). Oznacza to, że oszczędność może sięgać 250 000 bajtów, a nawet więcej.

Strategia ta nie wyczerpuje możliwości kompresji w odniesieniu do języka polskiego. Wiadomo, że język ten jest silnie fleksyjny, przy czym wiele słów ma wspólne końcówki dla deklinacji lub koniunkcji. Dlatego powyższą strategię poszerzono o interpretator końcówek: w podstawowym słowniku pamięta się tylko odpowiedni fragment tematu oraz odsyłacz do „macierzy" końcówek słowa. W ten sposób, na przykład, można w siedmiu bajtach słownika zapamiętać dziewiętnaście wariantów deklinacyjnych słowa „fizyka" i „fizyczny".

Generowanie hipotez słów podobnych

Jeśli dane słowo nie zostało znalezione w słowniku, to może to oznaczać, że:
- słowo jest poprawne, ale nie znane systemowi,
- użytkownik popełnił błąd przy pisaniu (nieumyślnie nacisnął inny klawisz, dodatkowy klawisz lub opuścił literę),
- użytkownik przestawił litery,
- użytkownik popełnił błąd ortograficzny.

Oczywiście w pierwszej sytuacji jedynie użytkownik może podjąć poprawną decyzję (zignorować słowo lub dodać je do słownika). W pozostałych – Medytor próbuje pomóc użytkownikowi i na życzenie generuje pięć możliwych hipotez słów podobnych. Z uwagi na docelową obszerność słownika, techniki obliczania odległości [4] od kolejnych

słów słownika wydają się być zbyt czasochłonne. Medytor opiera się na metodzie prób i błędów: generuje wiarygodny wariant słowa i porównuje go ze słownikiem. W pierwszej kolejności zakłada, że użytkownik nacisnął o jeden klawisz za dużo: np. dla słowa *dorbry* generuje hipotezy: *orbry, drbry, dobry* itd. a następnie sprawdza ich istnienie w słowniku. Potem sprawdza zamianę dwóch kolejnych liter („czeski błąd"); następnie zakłada, iż użytkownik popełnił błąd ortograficzny (np. *ż* zamiast *rz*). Po wyczerpaniu tych możliwości przechodzi do sprawdzenia hipotezy omyłkowo naciśniętego klawisza, a po niej – hipotezy o opuszczeniu litery. Proces ten jest przerywany w chwili znalezienia pięciu kandydatów. Kolejność poszczególnych rodzajów generowania hipotez jest uzasadniona przede wszystkim ich kosztem obliczeniowym.

Polskie znaki

Zastosowanie polskich znaków narodowych wymaga ustalenia:
- ich reprezentacji wewnętrznej,
- reprezentacji na ekranie monitora,
- reprezentacji na drukarce,
- sposobu wprowadzania z klawiatury,
- sposobu komunikacji z innymi systemami (np. z dBase w wypadku pobierania danych do tekstów parametryzowanych).

Wybór wewnętrznego kodu reprezentacji polskich znaków nie może być dowolny. Oprócz spełnienia oczywistych wymagań, takich jak możliwość realizacji sprzętowej na drukarce lub monitorze, niekolidowanie ze znakami sterującymi drukarki, należy zapewnić możliwość alfabetycznego sortowania. Dla szesnastu polskich liter *ą, ć, ę, ł, ń, ó, ś, ż, i Ą, Ć, Ę, Ł, Ń, Ó, Ś, Ż* przyjęto następującą konwencję: kod polskiej litery = kod ASCII litery łacińskiej + 80*H*; dodatkowo $\dot{z} = z + 81H$ oraz $\dot{Z} = z + 81H$. W systemie SYS8688 nazwano ten kod NPN-POL. Wynikający z tej konwencji sposób sortowania słów został użyty w poniższym podprogramie porównania leksykograficznego napisanym w języku C, a odpowiadającym bibliotecznej funkcji standardowej *strcmp* (*char *, char **):

```
int. pol_strcmp (pierwszy_tekst, drugi_tekst)
        unsigned char *pierwszy_tekst, *drugi_tekst);



(* Funkcja zwraca 0, gdy oba ciągi znaków są identyczne,
(*                liczbę ujemną, gdy ciąg pierwszy
(*                    leksykograficznie poprzedza drugi
(*                liczbę dodatnią, gdy ciąg drugi
(*                    leksykograficznie poprzedza pierwszy
*)
```

```
unsigned char *pp, *pd; int cp, cd;
for (pp = pierwszy_tekst, pd = drugi_tekst
    ; (*pp|0x80)*pp == *pd; pp++, pd ++);
if (cp = *pp & 0x7f) == (cd = *pd & 0x7f) return 0; /* identyczne */
if (cp&0x7f)>(cd&0x7f) return cp & 0x7f-cd & 0x7f;
return     cp-cd;
```

Polskie litery wprowadza się w konwencji stosowanej przez wielu użytkowników edytora Chiwriter, tj. przez naciśnięcie klawisza F_2, a następnie odpowiedniego klawisza litery łacińskiej; dla *ż* trzeba nacisnąć F_2 i *x*, a dla $\dot{Z}$ – F_2 i *X*.

W komunikacji z innymi systemami zastosowano łatwy do zapamiętania kod dwuznakowy używany w pakiecie TMK: litera polska = litera łacińska (dla 16 znaków), a dla *ż* i $\dot{Z}$ odpowiednio $\dot{z} = z$ ⓐ, $\dot{Z} = Z$ ⓐ (np. Żółkiewski = Z ⓐ o^ l^kiewski. Kod ten nazwano NPN–ASCII.

Polskie znaki na monitorze i drukarce zaimplementowano w sposób elastyczny. Z jednej strony polski tekst musi być jednoznacznie identyfikowany na monitorze i na drukarce przy użyciu standardowego sprzętu, z drugiej zaś musi istnieć możliwość ewentualnego wykorzystania sprzętowej implementacji polskich znaków.

Oczywistym rozwiązaniem jest użycie trybu graficznego monitora (i drukarki). W tym wariancie pracy znaki graficzne są zgodne z kodem NPN–POL. Znacznie szybszy jest jednak tryb tekstowy. Na ekranie stosuje się standardową reprezentację litera polska = litera łacińska. W celu osiągnięcia jednoznaczności polskie litery (16 znaków) są opcjonalnie pokazane z atrybutem UNDERLINE, a litery *Ż* i *ż* mają dodatkowo atrybut BLINK. Na drukarce stosuje się technikę nadruku:

ą = a + ˛	ę = e + ˛	ó = o + ´	ł = l + /	ń = n + ´
Ą = A + ˛	Ę = E + ˛	Ó = O + ´	Ł = L + /	Ń = N + ´
ć = c + ´	ś = s + ´	ż = z + ˙	ż = z + i	
Ć = C + ´	Ś = S + ´	Ż = Z + ˙	Ż = Z + –	

Powyższa konwencja reprezentacji nie jest zaprogramowana na stałe. Specjalnym programem ZNAK można dokonać przedefiniowania przyporządkowania: polski znak – znak monitora lub drukarki, np. na kod Mazovii, DHN, CSK lub inny dowolny, niezależnie dla drukarki oraz dla monitora, przy czym sposób wewnętrznego kodowania i sortowania nie ulegnie żadnej zmianie.

* * *

Na obecnym etapie rozwoju zastosowań informatyki omówione możliwości Medytora wydają się stanowić minimum wymagań, jakie należy stawiać pojawiającym się nowym edytorom tekstów. Nie można oczekiwać, by powyższe warunki mogła spełnić adaptacja programów pochodzenia zagranicznego – faktycznie polski edytor może powstać jedynie w kraju.

Kolejnym etapem rozwoju powinno być wyposażenie edytorów w mechanizmy kontroli poprawności syntaktycznej, a w przyszłości być może – stylistycznej.

Należy zwrócić uwagę na nową jakość w sformułowaniu zadań tekstu w systemie komputerowym, a także edytora takiego tekstu. W klasycznym ujęciu edytor przygotowuje „skostniały" tekst przeznaczony do wydruku lub użytkowania w innej formie raz ustalonego tekstu. Oprogramowanie wokół Medytora służy „ożywieniu" tekstu. Tekst przygotowany przez Medytor „przejmuje" (dzięki podsystemowi warunkowego generowania i parametryzowania tekstu) część prostych funkcji przetwarzania danych macierzystego systemu, umożliwiając oddzielenie części zalgorytmizowanej zagadnienia od opisowej. Tekst przygotowany przez Medytor może mieć wręcz charakter prostego systemu ekspertowego, opartego na metodzie wnioskowania *„forward–chaining"* – dzięki możliwości zapisu reguł wnioskowania w postaci warunkowego generowania tekstów, przy czym generowanym tekstem byłoby przypisanie wartości lokalnym parametrom Medytora.

Konsekwencją konceptualnego rozwoju systemu Medytor będzie w przyszłości ściślejsze nawiązanie do trendu systemów typu HYPERTEXT, w których następuje odwrócenie ról systemu macierzystego i systemu objaśnień Help. Tradycyjnie podsystemy Help pełnią funkcję usługową wobec systemów macierzystych. W systemie HYPERTEXT wiodącą rolę odgrywa aktywny tekst objaśnień, odwołujący się do usług systemów baz danych, systemów ekspertowych, klasycznych zalgorytmizowanych prezentacji audio-video i innych. System Medytor spełnia niektóre elementarne funkcje systemu HYPERTEXT, m.in. umożliwiając odwoływanie się z jednego tekstu objaśnień do innego (zawierającego np. szczegółowe objaśnienia pojęć używanych w tekście podstawowym, których definicje w tekście podstawowym byłyby zbędne).

LITERATURA

[1] Algebra C. N.: String similarity and misspellings. Comm. ACM, Vol. 10, No. 5, pp. 302–313, 1967

[2] Kłopotek M.: TMK – pakiet syntezy mowy dla IBM PC. Informatyka, nr 1, 1989

[3] Kłopotek M., et al.: SYS8688 – system pozyskiwania i weryfikacji wiedzy medycznej. Informatyka, nr 2, 3, 1990

[4] Lamprecht H.: Worterkennung auf der Basis partieller Uebereinstimmung von Wortfragmenten. INFORMATIK, No. 32, pp. 32–34, 1985.

Jeśli brakuje Państwu jakiegoś numeru INFORMATYKI z ubiegłego okresu, proszę do nas napisać lub zadzwonić (39-14-34) – wyślemy egzemplarze za zaliczeniem pocztowym.

ZDZISŁAW WRZESZCZ
Instytut Maszyn Matematycznych
Warszawa

Język Lisp i elementy sztucznej inteligencji (2)

Konstruowanie funkcji i programu

Ważną, często podkreślaną cechą języka Lisp jest to, że jedynymi elementami syntaktycznymi są wyrażenia symboliczne, czyli *s*-wyrażenia, z którymi zapoznaliśmy się w poprzedniej części artykułu. Tak więc zarówno program, jak i dane przyjmują postać *s*-wyrażenia. Jeśli *s*-wyrażenie jest listą, to jest ono traktowane przez interpreter jako forma funkcjonalna. Zachęcająco można więc powiedzieć, że pisanie programu lispowego oznacza tworzenie takich form funkcjonalnych, które mogą być zinterpretowane przez Lisp. Służy do tego przede wszystkim zestaw funkcji systemowych (wbudowanych); niektóre z nich poznaliśmy w części pierwszej. W części drugiej skoncentrujemy uwagę na pewnych specjalnych formach funkcjonalnych niezbędnych do tworzenia programu.

FORMY FUNKCJONALNE

Jak wiadomo, program podawany do interpretacji przez Lisp jest traktowany jako pewna forma funkcjonalna, tj. lista, której pierwszym elementem jest nazwa funkcji, zaś pozostałe elementy listy to argumenty tej funkcji. Lisp dokonuje wartościowania takiej funkcji. Jeśli funkcja należy do pewnego repertuaru podstawowego wartościowaniu są poddawane wszystkie jej argumenty, a liczba argumentów jest stała, to mamy do czynienia ze zwykłą formą funkcjonalną. Każdy interpreter Lispu dysponuje pewnym podstawowym zestawem definicji funkcji dla zwykłych form funkcjonalnych: zestaw taki obejmuje co najmniej następujące grupy funkcji:

- funkcje manipulowania zawartością listy, np. CAR,
- funkcje sterujące wykonaniem programu, np. COND,
- predykaty, np. GRATERP,
- funkcje arytmetyczne, np. +, −, *, /,
- funkcje wejścia i wyjścia, np. READ, PRINT,
- funkcje logiczne: AND, OR, NOT.

Formy specjalne

Specjalnymi, nazywane są takie formy funkcjonalne, które
a) przyjmują zmienną liczbę argumentów lub
b) w których argumenty są traktowane inaczej niż w formie zwykłej.

Przykładem dla sytuacji (a) jest funkcja MAX, która z dowolnej liczby argumentów liczbowych wybiera wartość największą. Przykładami formy (b) są: QUOTE, a także funkcja LIST i SETQ, w których argument nie jest wartościowany.

Funkcja COND

Zapoznamy się teraz z pewną formą specjalną, szczególnie użyteczną w sterowaniu wykonaniem programu lispowego. Jest to funkcja COND, która ma następującą postać ogólną:

(COND (*w*1 *r*1) (*w*2 *r*2)... (*wn rn*)

Lista argumentów jest tu dowolna, argumenty mają postać klauzul i składają się z dwóch wyrażeń: warunku *wi*, $i = 1..$ n oraz rezultatu *ri*, $i = 1..$ n. Wartością funkcji COND jest ten rezultat *rk*, dla którego warunek *wk* jest spełniony. Jeśli żaden z warunków nie zostanie spełniony, to rezultat jest nieokreślony. Stąd też, zgodnie z ogólną praktyką programowania, w miejsce ostatniego warunku *wn* jest wstawiana wartość T.

DEFINIOWANIE FUNKCJI

Funkcja DEFUN

Aby określić nową funkcję spoza zestawu podstawowego, programista musi podać definicję tej funkcji. Definicja funkcji jest następującą formą:

(DEFUN „*nazwa-funkcji*" („*pf*1"... „*pfn*") „*w*1"... „*wm*")

Symbole objęte znakami cudzysłowu nie są wartościowane przez Lisp po wprowadzeniu funkcji DEFUN, jest ona więc formą specjalną. Znaczenie argumentów funkcji DEFUN jest następujące:

- argument pierwszy jest to dowolny literał oznaczający nazwę definiowanej funkcji,
- argumentem drugim jest lista parametrów formalnych *pf*1, ...*pfn*, które będą związane z wartościami argumentów funkcji wywołującej o tej samej nazwie,
- argument trzeci i ewentualnie następne, to sekwencja *s*-wyrażeń *w*1, ... *wm*, tworzących ciało funkcji.

Po wprowadzeniu funkcji Lisp zwróci symbol NAZWA-FUNKCJI jako wartość funkcji DEFUN. Wprowadzenie formy

»(*nazwa-funkcji s*1... *sn*)

powoduje, że wartość wyrażeń *s*1... *sn* są dowiązane do parametrów formalnych *pf*1... *pfn* i dla tych wartości następuje wartościowanie *s*-wyrażeń *w*1... *wm* ciała funkcji DEFUN. Wartością funkcji NAZWA-FUNKCJI jest wartość wyrażenia *wm*. Rozpatrzmy prosty przykład funkcji mnożącej dowolną wartość x przez liczbę 5.

»(defun mnoz-przez-piec (x) (* 5x))

MNOZ-PRZEZ-PIEC

Wywołanie formy

»(mnoz-przez-piec 10)
50

jest interpretowane przez Lisp następująco:

- wartościowany jest argument formy wywołującej 10, wynikiem jest sama liczba 10,
- przypisanie parametrowi formalnemu *x* wartości 10,
- wartościowanie ciała funkcji, czyli (* 5 10),
- podanie wyniku wartościowania, czyli 50.

FUNKCJA JAKO ARGUMENT

Niezmiernie pożyteczne okazało się przekazywanie funkcji jako argumentu. Rozpatrzmy następującą formę ogólną:

(FUNCALL *nazwa-funkcji arg*1... *argk*)

Pierwszy argument funkcji FUNCALL może zmieniać wartość zależnie od ustalenia. Jeśli ustalono, że

Do 10 listopada można jeszcze zaprenumerować INFORMATYKĘ na cały rok 1991

```
»(setq nazwa-funkcji '+)
```

to

```
»(funcall nazwa-funkcji 10 15)
25
```

Jeśli natomiast

```
»(setq nazwa-funkcji '*)
```

to wywołanie będzie następujące:

```
»(funcall nazwa-funkcji 10 15)
150
```

ZMIENNE GLOBALNE I LOKALNE

Symbol nie jest związany z żadną wartością, zanim nie zostanie wykonane przypisanie. Każdy symbol może mieć przypisaną jedną wartość globalną i dowolną liczbę wartości lokalnych, jednak w danym kontekście symbol reprezentuje co najwyżej jedną wartość.

Przypisania symbolowi wartości globalnej dokonuje się za pomocą funkcji SETQ np.:

```
»(setq z (* 2 3) )
6
```

W wyniku interpretacji nastąpi najpierw wartościowanie (* 2 3), a następnie przypisanie wartości 6 jako wartości globalnej symbolu z. Odzyskanie wartości globalnej realizuje się za pomocą funkcji

```
»(SYMBOL-VALUE 'z)
6
```

Możliwość tworzenia symboli z wartością globalną ogromnie ułatwia budowę programu lispowego dzięki zmiennym, które są dostępne dla różnych funkcji. Zaleca się jednak dużą ostrożność przy korzystaniu z wartości globalnej ze względu na funkcje o działaniu destrukcyjnym, co może spowodować nie kontrolowane działanie programu. Z tej racji w wielu przypadkach korzystniejsze może okazać się operowanie wartościami lokalnymi, zwłaszcza tam, gdzie dana informacja nie jest obiektem zainteresowania wielu funkcji w danym programie.

Sposoby ustalania lokalnych wartości są następujące:

- jako parametr formalny w definicji funkcji: jakiekolwiek odwołanie do symbolu z zachodzące w ciele funkcji odnosi się jedynie do wartości lokalnej z, która ma charakter „prywatnej własności" danej funkcji i jest nieosiągalna poza tą definicją;
- przez ustalenie symbolu jako zmiennej lokalnej w specjalnej formie funkcjonalnej LET i LET*.

Forma LET

Ogólna postać formy LET jest następująca:

(LET (*x*1 *w*1)... (*xk wk*)) *e*1...*ek*)

Symbole *x*1...*xk* są zmiennymi lokalnymi, które uzyskują wartości początkowe równe wartościom *w*1...*wk* po ewaluacji, *e*1...*ek* to składniki ciała formy LET.

PROGRAM ITERACYJNY

Umiemy już manipulować zawartością listy, dysponujemy silnym zestawem funkcji, umiemy też tworzyć nowe funkcje. Okazuje się jednak, że nie wystarcza to do efektywnego konstruowania programów. Przykładowo, aby listę o znacznej liczbie elementów przetworzyć za pomocą funkcji Fu, środki dotychczas omówione wymagają wielokrotnego wywoływania funkcji Fu, proporcjonalnie do długości listy. Niezbędny jest więc pewien mechanizm, który raz wprowadzony będzie powodować wykonywanie tej funkcji tyle razy, ile będzie potrzeba.

Funkcja PROG

Forma ta, dziś zarzucona przez programistów lispowych, ma następującą postać ogólną:

(PROG (*X*1... *Xk*) *E*1... *Em*)

Zmiennym lokalnym *X*1... *Xk* są przypisane wartości początkowe NIL; *E*1... *Em* – są dowolnymi *s*-wyrażeniami: jeśli *Ei* jest symbolem to pełni rolę znacznika nie podlegającego wartościowaniu. Forma PROG występuje najczęściej wewnątrz definicji funkcji DEFUN. Przykładem niech będzie proste przetworzenie tekstu. Z danego tekstu brane jest pierwsze słowo i dodane na początek odpowiedzi:

```
»(defun pozdr-komp (x)
  (prog (y)
    (setq y (list (car x) '(panie kapitanie) ) ) (print y) ) )
POZDR-KOMP
»(setq x '(czołem żołnierze) )
CZOŁEM ŻOŁNIERZE
»(pozdr-komp x)
(CZOŁEM (PANIE KAPITANIE) )
NIL
```

Zwróćmy uwagę na podwójny charakter zmiennej x: jest ona zmienną globalną w wywołaniu funkcji POZDR-KOMP i zmienną lokalną w definicji tej funkcji.

Tworzenie pętli

Formę PROG można przetworzyć w formę iteracyjną po wprowadzeniu:

- funkcji (GO *znacznik*),
- symbolu *znacznik*,
- warunku powtórzeń,
- funkcji powrotu (RETURN *s-wyrażenie*).

Rozpatrzmy przykład. Oddzielnym pakietom a, b, c, d są nadane imiona odbiorców:

```
»(setq a 'jan)
JAN
»(setq b 'adam)
ADAM
»(setq c 'ewa)
EWA
»(setq d 'jola
JOLA
```

Utworzono następnie jeden pakiet:

```
»(setq pakiet '(a b c d) )
(A B C D)
```

którego zawartość może być powiększana w miarę potrzeby. Chcąc zapoznać się z zawartością pakietu utworzymy funkcję:

```
»(defun zawartość (pa)
  (prog ( (x pa) (y nil) )
petla
    (cond (null x) (return 'komplet) ) )
      (setq y (car x) )
      (print (eval y) )
      (setq x (cdr x) )
      (go petla) ) )
```

ZAWARTOŚĆ
a następnie z niej skorzystamy:

```
»(zawartość pakiet)
JAN
ADAM
EWA
JOLA
KOMPLET
```

A oto objaśnienie wyrażeń funkcji ZAWARTOŚĆ:
x, y – zmienne lokalne o wartości początkowej pa i nil;
petla – symbol znacznika pętli iteracyjnej;
(null x) – wyrażenie warunku;
(return 'komplet) – wyrażenie wartościowane, jego wartość jest zwracana jeśli wyrażenie warunku jest spełnione;
(setq y (car x)) – pobranie pierwszego elementu z pakietu;
(print (eval y)) – wydrukowanie aktualnej zawartości pakietu;
eval – funkcja wartościująca, dostarcza wartość przypisaną danemu symbolowi, elementowi listy PAKIET;
(setq x (cdr x)) – redukcja listy pakietu o element pierwszy na liście;
(go petla – funkcja sterująca nakazująca powrót do znacznika PETLA.

Funkcja DO

Funkcja DO odgrywa podobną rolę, jak wyrażenie WHILE w języku klasycznym (np. w Pascalu). Funkcja ta pozwala formułować działanie iteracyjne z pominięciem wyrażeń o nawrocie, występujących explicite w konstrukcji PROG. Wyrażenie typu DO ma następującą postać ogólną:

(DO ((X1 V01 *VR*1)... (X*k* V0*k* V*Rk*)) (warunek *E*) *E*1... *Em*)

gdzie:
X1 ...X*k* – zmienne lokalne,
V01 ... V0*k* – *s*-wyrażenia, których wartość staje się wartością początkową zmiennych lokalnych,
V*R*1 ... V*Rk* – *s*-wyrażenia, których wartość staje się wartością zmiennych po kolejnym kroku iteracyjnym,
(*warunek E*) – klauzula warunku: po spełnieniu warunku *warunek* wartościowej podlega wyrażenie *E*, a rezultat jest wartością funkcji DO,
*E*1 ... *Em* – wyrażenia wykonane w każdym kroku iteracji.

Przykładem zastosowania DO może być algorytm mnożenia dwóch liczb, działających na zasadzie dodawania jedynki do jednej z liczb

```
»(defun mnoz-przez-dod (x1 x2)
  (do ( (wynik 0 (+ wynik x1) ) (licznik 0 (+ 1 licznik) ) )
    ( (equal licznik x2) wynik) ) )
MNOZ-PRZEZ-DOD
```

A teraz wykonajmy mnożenie dwóch liczb, np. 6 i 15:

```
»(mnoz-przez-dod 6 15)
90
```

PROGRAM REKURSYJNY

Rekursja podobnie jak iteracja, zawiera powtórzenie jako główny mechanizm wykonania programu. Ponadto konstrukcja takiego programu wymaga jednak:

- przeformułowania danego problemu do postaci prostszej, lecz zgodnej z formą problemu,
- określenia sposobu wykorzystania prostszej postaci problemu,
- określenia pewnej postaci „oporowej", która nie wymaga dalszej dekompozycji i może być wykonana przez Lisp bezpośrednio,
- określenia kryterium pozwalającego stwierdzić, że osiągnięto postać oporową problemu.

Popularnym zadaniem do demonstracji rekursji jest sumowanie liczb od 1 do n. Zwróćmy uwagę, że wyrażenie (suma-do 8) ma tę samą wartość co (+ 8 (suma-do 7)). Podobnie jest w przypadku kolejnych liczb mniejszych. Wartością oporową jest 0. Cały program zapiszemy więc następująco:

```
»(defun suma-do (n)
  (cond ( ( (zerop n) 0)
    (t (+ n(suma-do (1 - n) ) ) ) ) ) )
SUMA-DO
```

Sprawdźmy więc, jak to działa:

```
»(suma-do 10 )
55 ;;; Zgadza się!
```

WEJŚCIE I WYJŚCIE

Mówi się często, że o praktycznej przydatności języka decydują jego operatorzy wejścia i wyjścia; Lisp jest pod tym względem również bardzo dobry. Wprowadzanie i wyprowadzanie danych do programu lispowego jest wykonywane z pomocą funkcji, podobnie jak wszystko inne. Podstawową formą danych jest *s*-wyrażenie i typy od niego pochodne, są jednak także nowe typy.

Funkcja READ

Funkcja READ jest bezargumentowa. Gdy interpreter spotyka formę (READ), to zawiesza swoje działanie oczekując podania z terminala jednego s-wyrażenia. Po wprowadzeniu tego wyrażenia Lisp wypisze to wyrażenie jako wartość funkcji READ. Przykładowo:

```
»(setq z (read) ) (x1 x2 x3)
(X1 X2 X3)
»z
(X1 X2 X3)
```

Drugie wyrażenie w pierwszym wierszu zostało podane przez operatora, co wyraźnie nie było oznaczone funkcją QUOTE. Potrzeby takiej nie ma, gdyż interpreter nie wartościuje wyrażenia uzyskanego po funkcji READ. Po otrzymaniu wyrażenia (x1 x2 x3) Lisp przypisał je symbolowi z, o czym powiadomił wpisując uzyskane wyrażenie (drugi wiersz). Można też sprawdzić wartość z, co również uczyniliśmy.

Funkcja PRINT

Jest to funkcja jednoargumentowa. Funkcja ta powoduje wydruk swojego argumentu. Przykład:

```
»(defun czyta-wykona-drukuje ( )
(prog ( (x (read) ) (z 0) )
(print '(podales liczbe) ) (print x)
(setq z (* 13 x) ) (print z) ) )
CZYTA-WYKONA-DRUKUJE
```

Popatrzmy teraz, jak to przebiega:

```
»(czyta-wykona-drukuje) 5
(PODALES LICZBE)
5
65
NIL
```

Charakterystyczne cechy funkcji PRINT są następujące:

- wydruk rozpoczyna od nowego wiersza,
- wydruk następuje po odstępie jednego wiersza.

Funkcja PRIN1

Funkcja drukuje wyłącznie dany obiekt, bez przejścia do nowego wiersza.

Funkcja PRINC

Funkcja działa podobnie jak PRIN1, jednak jej argument nie musi być *s*-wyrażeniem. Wobec tego funkcja doskonale nadaje się do drukowania tekstów, komunikatów systemu użytkowego itp.

Dane typu sznur

Sznur jest nowym typem danych, różnym od typów związanych z *s*-wyrażeniem. Sznur jest to sekwencja znaków otoczona cudzysłowem.

Wartością danej typu sznur jest ona sama. Przykład:

```
»„funkcja"
„funkcja"
»„dzień dobry"
„dzień dobry"
```

Funkcja PRINT drukuje dane sznurowe wraz z cudzysłowem, natomiast funkcja PRINC drukuje takie dane bez cudzysłowu, czyli tak,

jak byśmy to chcieli widzieć w druku. Przykładem niech będzie następująca procedura notowania czasu:

```
»(defun podaj-czas ( )
(prog (godz min)
(princ „podaj godzine") (setq godz (read) )
(princ „podaj minuty") (setq min (read) )
(terpri) (princ „Zanotowany czas:")
(prin1 godz) (princ „:") (prin1 min) ) )
PODAJ CZAS
```

Wywołanie tej procedury:

```
»(podaj-czas) Podaj godzine 9
podaj minuty 39

Zanotowany czas : 9:39
```

Zmienne typu strumień

Common **LISP** dysponuje typem danych o nazwie strumień (ang. *stream*). Strumień pełni rolę kanału pośredniczącego między programem a jego otoczeniem, tj. urządzeniem lub plikiem. Zmienna taka staje się wejściem i wyjściem dla pliku. Zmienne typu strumień umożliwiają pisanie programu lispowego niezależnie od sprzętu – wejście i wyjście tego programu definiuje się jako wejście – wyjście do kanału.

Pisanie do pliku i czytanie z pliku

Dany jest plik o nazwie np. a1.lsp, do którego zamierzamy pisać dane. Tworzymy więc strumień o nazwie np. kanal-wy. Przed operacją pisania strumień należy otworzyć poleceniem OPEN:

```
»(setq kanal-wy (open „al" : direction : output))
#⟨A1.LSP⟩
```

Funkcja OPEN ma dwa argumenty, z których pierwszy to nazwa pliku, a dwa pozostałe oznaczają kierunek przesyłania. Symbole poprzedzane dwukropkiem są słowami kluczowymi. W wyniku interpretacji wyrażenia, system wydaje napis zawierający np. nazwę pliku w nawiasach ostrych poprzedzoną znakiem #, napis ten nie jest objęty standardem.

Zapisu do pliku dokonuje się za pomocą jednej z funkcji pisania, np. PRINT, dodając zmienną strumieniową jako dodatkowy argument:

```
»(print '(aa bb cc) kanal-wy)
(AA BB CC)
»(print 'adam kanal-wy)
ADAM
```

Po operacji pisania należy kanał zamknąć, co wykonujemy za pomocą funkcji CLOSE:

```
»(close kanal-wy)
NIL
```

Aby teraz odczytać plik a 1.lsp, w tym celu należy otworzyć kanał *kanal-we* oraz użyć funkcji READ z parametrem, którym jest nazwa kanału *kanal-we*:

```
»(setq kanal-we (open „a1.lsp" : direction : input)
#⟨A1.LSP⟩
»(read kanal-we)
(AA BB CC)
»(read kanal-we)
ADAM
»(close kanal-we)
NIL
```

Po przeczytaniu zawartości pliku kanał został zamknięty. Wykonano tylko dwa odczyty, gdyż plik zawierał tylko dwie dane; trzecia próba odczytu skończyłaby się błędem zgłoszonym przez Lisp.

* * *

W artykule przedstawiono główne formy funkcjonalne potrzebne do konstruowania programu w języku Common **LISP**. Program taki tworzy się jako pewien zbiór definicji funkcji. Każda z tych definicji może zawierać odwołania od innych funkcji (zagnieżdżanie), co pozwala budować hierarchiczną strukturę modułów programu; struktura taka może w pełni odpowiadać strukturze rozwiązywanego problemu. Przedstawione tu elementy języka Common **LISP** ukazują jego możliwości zarówno w programowaniu zadań na danych symbolicznych, jak też i w programach numerycznych; Common **LISP** to nowoczesny język uniwersalnego zastosowania. Zasób podanych tu wiadomości o tym języku powinien być wystarczający do czytania tekstu lispowego. Jednak z chwilą przejścia do aktywnego posługiwania się językiem, tj. do formułowania i realizacji programów za pomocą określonego translatora, zajdzie potrzeba skorzystania z podręczników i dokumentacji firmowej.

Konferencje

II Wiosenna Beskidzka Szkoła PTI

Górnośląski Oddział PTI zorganizował w dniach 2–4 maja br. w Szczyrku Wiosenną Beskidzką Szkołę **„Obszary Zastosowań Informatyki"**. Charakter szkoły jest świadomie różny od dwóch innych Szkół PTI: mrągowskiej i świnoujskiej. Celem organizatorów jest przedstawienie praktykującym programistom (głównie z Polski południowej) informacji o istniejących możliwościach poszerzenia warsztatu zawodowego.

Szkoła Beskidzka miała następujący przebieg.

● W dniu 2 maja odbyły się wykłady **Joanny Kurzok** z ISS Katowice o standardach graficznych; **Andrzeja Kwietnia** z Politechniki Śląskiej na temat praktycznych doświadczeń z komputerowego przetwarzania obrazów oraz **Leszka Luchowskiego** z IITiS PAN z Gliwic o systemie analizy obrazów i metodach prezentacji obrazów w systemach wizyjnych. W sesji popołudniowej **Piotr Fuglewicz** opowiedział o komputerowo realizowanej statystyce językoznawczej.

● Dzień 3 maja poświęcony był sieciom komputerowym w aspekcie ich praktycznego wykorzystania. **Józef Maroński** z IPI PAN przedstawił informacje o protokole X.400, zastosowaniach poczty elektronicznej, stanie rozwoju sieci POLPAK, systemie EDIFACT i wielu innych problemach związanych z sieciami. **Aleksandra Bojdak** z ISS scharakteryzowała systemy MAP i TOP oraz podała przykład aplikacji z tego obszaru, realizowanej w ISS. Dzień zamknął **Jan Stożek** wykładem o sieci FidoNet – amatorskiej sieci o zasięgu globalnym. Próba połączenia się z siecią przez węzeł zainstalowany w redakcji miesięcznika KOMPUTER nie powiodła się z uwagi na powszechnie znany stan sieci telekomunikacyjnej w Polsce. Uzupełnieniem tematyki sieciowej była informacja przedstawiciela firmy Soft-tronik (wraz z firmą BASF współsponsora imprezy) na temat oprogramowania sieciowego wykorzystywanego w tej firmie.

● Dzień 4 maja poświęcono systemom czasu rzeczywistego. **Krzysztof Nałęcki** z Politechniki Śląskiej opowiedział, kiedy system można nazwać systemem czasu rzeczywistego i co to naprawdę znaczy. **Wojciech Trojniar** z IITiS PAN przedstawił praktyczne realizacje i problemy związane z tworzeniem systemów czasu rzeczywistego. **Marek Średniawa** z Politechniki Warszawskiej zaprezentował metodologię tworzenia systemów czasu rzeczywistego przy użyciu narzędzia opartego na specyfikacji w języku SDL, Paweł **Gizbert-Studnicki** z UJ omówił analogiczne narzędzie zbudowane na podstawie języka Smalltalk. Sesję uzupełnił komunikat **T. Tomalskiego** o systemie sterowania załadunkiem miału węglowego.

Uczestnicy Szkoły sprawiali wrażenie zadowolonych. Frekwencja na wykładach, mimo pięknej pogody, była bardzo duża. Prezentacje programów związanych z wykładami, na jednym z czterech udostępnionych uczestnikom komputerów, przeciągały się do późnej nocy. Programy, oddane w powszechne użytkowanie, każdy z uczestników mógł przegrać na dyskietkę podarowaną przez firmę BASF. Materiały konferencyjne zostały wydane w trzech odrębnych zeszytach, które można jeszcze kupić po cenie kosztów w **Górnośląskim Oddziale PTI; 40-014 Katowice; ul. Mariacka 6, tel. (Katowice) 538-102.** Również pod tym adresem można już rezerwować miejsca na przyszłoroczną Szkołę, która odbędzie się również w Szczyrku, w dniach 7–10 maja 1991 i będzie prawdopodobnie poświęcona w całości warsztatowi pracy programisty (sieci i wielodostęp, bazy danych, narzędzia programowe). Organizatorzy proszą również o kontakt wszystkich, którzy chcieliby być wykładowcami szkoły o takiej tematyce.

P.F.

Kto nie zdążył zaprenumerować INFORMATYKI na cały rok przyszły – a chciałby ją otrzymywać – może zwrócić się do redakcji, wysyłamy egzemplarze za zaliczeniem pocztowym.

MEDCOM

ZASILANIE AWARYJNE

ZASILACZE BEZPRZERWOWE

naszej produkcji

UPS – 450

produkcji tajwańskiej

UPS–500

produkcji japońskiej

GENERATORY PRĄDOTWÓRCZE

różnych mocy

EO 3 90

Biuro
Al. Ujazdowskie 26/39
00–478 Warszawa tel. 28 93 57, teleks 81 70 60

ZAKŁAD PRODUKCYJNO-USŁUGOWY
ul. Wołodyjowskiego 41
02–724 Warszawa tel.43 04 41

MIŁOSZ MUSZYŃSKI
Instytut Informatyki, Automatyki i Robotyki
Politechnika Poznańska

Fizyczne struktury systemów baz danych (3)

Wielowymiarowe struktury danych

W większości zastosowań systemów baz danych trzeba zapewnić efektywny dostęp do danych według wielu atrybutów. Jest to szczególnie istotne w interakcyjnych systemach baz danych, w graficznych bazach danych, w systemach komputerowo wspomaganego projektowania, w systemach robotyki, wizji komputerowej itd.

Przy omawianiu wielowymiarowych struktur danych należy wprowadzić pojęcie zapytania częściowego. Jest to szczególny przypadek zapytania z zakresem, w którym zakresy dla pewnej liczby atrybutów obejmują całe dziedziny tych atrybutów, dla pozostałych atrybutów natomiast obejmują ściśle wyznaczone punkty. Przypadek taki można również interpretować jako zapytanie punktowe z brakującymi specyfikacjami niektórych wartości atrybutów. Zapytanie częściowe występuje bardzo często w praktyce i z tego powodu efektywność jego realizacji jest zazwyczaj rozpatrywana oddzielnie.

W ostatniej części artykułu omówiono możliwości realizacji dostępu według wielu atrybutów w bazach danych opartych na typowych, jednowymiarowych organizacjach plików dyskowych.

Dostęp do danych według wielu atrybutów z wykorzystaniem jednowymiarowych organizacji plików dyskowych

W jednowymiarowych organizacjach plików jeden wybrany atrybut jest atrybutem podstawowym i zazwyczaj wyznacza fizyczne uporządkowanie rekordów w pliku.

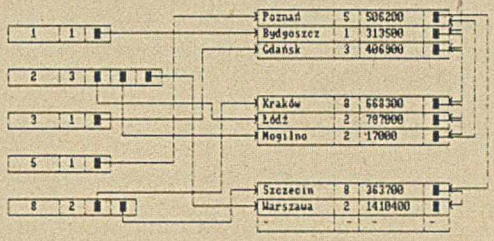
Indeks utworzony według atrybutu będącego podstawą uporządkowania fizycznego nazywa się **indeksem pierwotnym**, a atrybut z nim związany – **atrybutem pierwotnym**. Jest oczywiście możliwe utworzenie wielu indeksów według innych atrybutów; indeksy takie oraz atrybuty będące ich podstawą noszą nazwę **wtórnych**.

Dla zapytań z zakresem najefektywniejszy jest dostęp do danych według indeksu pierwotnego. Wymaga on bowiem najmniejszej liczby operacji odczytu i zapisu bloków dyskowych. Rekordy o zbliżonej wartości atrybutu wtórnego są natomiast rozproszone w całym obszarze pliku dyskowego. Z tego powodu dla zapytań z zakresem efektywności dostępu według indeksu wtórnego jest dużo mniejsza.

Najistotniejszą wadą indeksów wtórnych nie jest jednak mniejsza efektywność dostępu (dla zapytań punktowych nie odbiega ona od efektywności dostępu według indeksu pierwotnego), lecz duży stopień skomplikowania pliku i w konsekwencji poważnie zwiększona złożoność operacji wstawiania i usuwania. Indeks wtórny jest zawsze realizowany jako indeks gęsty. Dla indeksów wtórnych nie istnieje bowiem żaden porządek, umożliwiający znalezienie jednego rekordu na podstawie informacji o lokalizacji innego rekordu. Plik dyskowy z wieloma gęstymi indeksami wtórnymi charakteryzuje się sztywną i skomplikowaną strukturą, w której wstawianie i usuwanie są operacjami złożonymi.

Innym rozwiązaniem stosowanym w celu zapewnienia dostępu do danych według wielu atrybutów jest tak zwany indeks odwrócony. Jest on złożony z listy wartości atrybutu wtórnego oraz wskaźników do odpowiadających tym wartościom rekordów (rys. 1). Do wad plików z indeksem odwróconym należą:
– trudność efektywnej realizacji wyszukiwania wartości w indeksie odwróconym,
– skomplikowana struktura pliku i w konsekwencji duża złożoność operacji wstawiania i usuwania.

Pierwsza z wymienionych wad jest konsekwencją złożonej struktury indeksu odwróconego. Indeks odwrócony stanowi zbiór rekordów o zmiennej długości, odpowiadającej liczbie rekordów o jednej wartości atrybutu wtórnego. Wyszukiwanie w indeksie odwróconym realizuje się czasem z użyciem trie-drze [6] (opisanych w pierwszej części artykułu). Druga z wymienionych wad wynika z przyczyn analogicznych, jak dla pliku z wieloma indeksami wtórnymi [5].



Rys. 1

Najbardziej efektywna strategia realizacji wieloatrybutowego zapytania punktowego w pliku o strukturze jednowymiarowej, wyposażonym w indeksy wtórne i odwrócone, jest następująca:
- wyszukanie wszystkich rekordów spełniających warunek równości dla jednego z atrybutów przy użyciu któregoś z indeksów;
- atrybut powinien być tak dobrany, aby liczba wyszukanych rekordów była minimalna;
- odrzucenie tych rekordów, które nie spełniają warunków zapytania ze względu na pozostałe atrybuty.

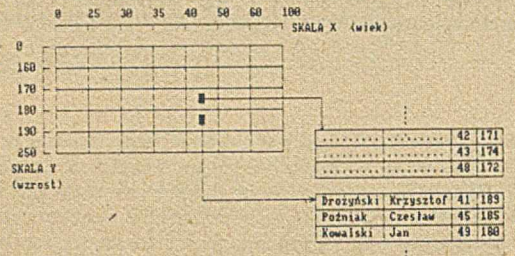
Widać, że nawet ta strategia jest związana z potencjalną koniecznością wykonania wielu niepotrzebnych odczytów bloków dyskowych. Z tego względu jednowymiarowe struktury danych, wyposażone w mechanizmy dostępu do danych według wielu różnych atrybutów, nie są zaliczane do wielowymiarowych struktur danych, nawet gdyby mechanizmy te były bardzo efektywne. Od struktur wielowymiarowych wymaga się, aby wieloatrybutowe zapytanie punktowe było realizowane bez konieczności dostępu do rekordów nie spełniających warunków wyspecyfikowanych w zapytaniu.

PLIKI KRATOWE

Pliki kratowe (ang. *grid file*) są propozycją wielowymiarowej organizacji pliku dyskowego umożliwiającej efektywną realizację wieloatrybutowych zapytań punktowych i zapytań z zakresem [7, 12]. Rekordy są reprezentowane jako punkty wielowymiarowej przestrzeni, w której każdy wymiar reprezentuje dziedzinę jednego atrybutu. Zakłada się, że liczba atrybutów wynosi k, przestrzeń rekordów jest więc k-wymiarowa. Każda z k osi tej przestrzeni jest podzielona na przedziały, wyznaczające podział przestrzeni na k-wymiarowe prostopadłościany. Podział ten jest dokonywany w taki sposób, aby wszystkie punkty zawarte w poszczególnych prostopadłościanach odpowiadały rekordom umieszczonym w jednym bloku dyskowym. Po przepełnieniu bloku dodaje się nowy przedział na dowolnie wybranej osi, przez co uzyskuje się nowy, drobniejszy podział przestrzeni rekordów. Liczba bloków pliku dyskowego jest równa lub mniejsza od liczby k-wymiarowych prostopadłościanów. Dwa prostopadłościany odpowiadające jednemu blokowi dyskowemu określa się mianem bliźniaczych. Powstawanie prostopadłościanów bliźniaczych jest skutkiem ubocznym podziału prostopadłościanów przepełnionych.

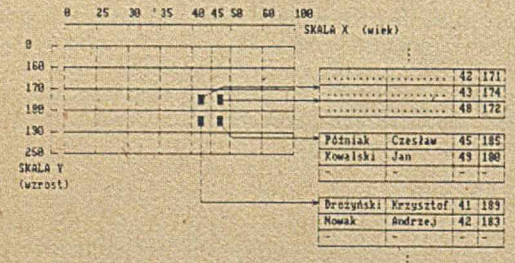
Opisana k-wymiarowa przestrzeń rekordów jest pojęciem abstrakcyjnym, reprezentowanym w pamięci w postaci tak zwanego katalogu.

Katalog pliku kratowego składa się z k-wymiarowej tablicy, zwanej tablicą kratową, oraz z k jednowymiarowych tablic, zwanych skalami. Tablica kratowa zawiera odwzorowanie między k-wymiarowymi prostopadłościanami a blokami dyskowymi. Każdy k-wymiarowy prostopadłościan jest reprezentowany przez element tablicy kratowej i zawiera numer odpowiadającego mu bloku dyskowego. Liczba elementów tablicy kratowej jest równa iloczynowi liczb przedziałów zdefiniowanych w skalach. Skala jest tablicą zawierającą liczby określające podział dziedzin atrybutów na przedziały.



Rys. 2

Na rysunku 2 przedstawiono przykładową strukturę pliku kratowego dla $k = 2$. Liczba przedziałów określonych przez skalę w osi X wynosi 7, a w osi Y wynosi 5. Tablica kratowa składa się więc z 35 elementów. Wyszukanie rekordów zawierających dane pracowników w wieku 45 lat charakteryzujących się wzrostem 185 cm wymaga wykonania następujących kroków:
– znalezienia numeru przedziału skali X, w którym znajduje się liczba 45; numer ten wynosi 5;
– znalezienia numeru przedziału skali Y, w którym znajduje się liczba 185; numer ten wynosi 4;
– odczytania numeru bloku dyskowego z tablicy kratowej G, numer ten wynosi G [5,4];
– odczytania bloku o numerze G[5, 4] i wyszukania w nim odpowiednich rekordów.
Realizacja wieloatrybutowego zapytania punktowego wymaga wykonania odczytu jedynie dwóch bloków dyskowych. W pierwszym bloku znajduje się katalog pliku kratowego, w drugim natomiast szukany rekord. Jeśli katalog pliku kratowego nie mieści się w jednym bloku, to tworzy się katalog dwu- lub nawet wielopoziomowy. Na poziomie najwyższym w tym katalogu znajduje się tak zwany katalog zgrubny; mieści się on w jednym bloku pamięci dyskowej i zawiera informacje o tym, w których blokach znajdują się odpowiednie części katalogu na bezpośrednio niższym poziomie (katalogu szczegółowego).

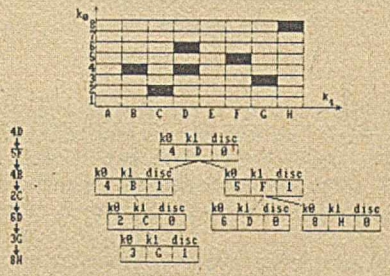


Rys. 3

Na rysunku 3 przedstawiono efekt takiej operacji wstawienia rekordu do pliku kratowego, która spowodowała przepełnienie bloku. Liczba zakresów wybranej skali została zwiększona, przez co zwiększyła się liczba elementów tablicy kratowej. Skutkiem ubocznym przepełnienia bloku jest powstanie elementów tablicy kratowej odpowiadających prostopadłościanom bliźniaczym.

k-d drzewa

Struktury zwane k-d drzewami zaproponowano jako rozszerzenie drzew binarnych, mające na celu umożliwienie obsługi dostępu do danych wielowymiarowych, a ściślej, k-wymiarowych [2]. W przeciwieństwie do węzła drzewa binarnego, który zawiera jedną wartość klucza oraz dwa wskaźniki do poddrzew tego węzła, węzeł k-d zawiera k wartości wszystkich atrybutów reprezentowanego przez ten węzeł rekordu, dwa wskaźniki do poddrzew węzła oraz specjalne pole dyskryminatora węzła. Dyskryminator węzła jest liczbą całkowitą z zakresu $\langle 0, k-1 \rangle$, określającą atrybut podstawowy węzła, czyli ten spośród k atrybutów k-d drzewa, który decyduje o wyborze lewego lub prawego poddrzewa podczas wyszukiwania oraz konstruowania drzewa. Główną właściwością atrybutu podstawowego jest to, że wszystkie wartości tego atrybutu w lewym poddrzewie są mniejsze, a w prawym poddrzewie są większe niż wartość atrybutu podstawowego. Wartości dyskryminatorów są wyznaczane w sposób cykliczny: dyskryminator korzenia jest równy zero, dyskryminatory bezpośrednich następników korzenia są równe jeden itd. Dyskryminatory następników węzłów o wartości dyskryminatora równej $k-1$ mają wartość równą zero. Wszystkie węzły k-d drzewa znajdujące się na tym samym poziomie mają jednakową wartość dyskryminatora (rys. 4).



Rys. 4

Wyszukiwanie w k-d drzewie przebiega analogicznie jak wyszukiwanie w drzewie binarnym i odbywa się przez porównywanie wartości atrybutów zadanych, jako argument poszukiwania z wartościami zawartymi w węzłach k-d drzewa. Przykładowo, wyszukanie rekordu, identyfikowanego przez klucz o wartości „3G" w 2-d drzewie pokazanym na rys. 4, składa się z następujących kroków:
– dyskryminator korzenia wynosi zero, porównuje się więc wartości atrybutu oznaczonego jako k_0; ponieważ $3 < 4$, więc do dalszego wyszukiwania wybiera się lewe poddrzewo,
– dyskryminator lewego następnika korzenia wynosi 1, porównuje się wartości atrybutu k_1; ponieważ G > B, więc wybiera się prawe poddrzewo,
– dyskryminator wynosi zero, $3 > 2$, więc wybiera się prawe poddrzewo,
– dyskryminator wynosi jeden, G = G, $3 = 3$, więc szukany rekord został znaleziony.

Powyższy przykład ilustruje proces realizacji zapytania punktowego, w którym wartości wszystkich k atrybutów przedstawiono jako argument poszukiwania. W przypadku zapytań częściowych, w których wyspecyfikowano tylko część spośród k wartości atrybutów, algorytm wyszukiwania jest bardziej skomplikowany. Wybór lewego lub prawego poddrzewa w węźle, którego dyskryminator odpowiada atrybutowi o niesprecyzowanej wartości, w ogólności nie jest możliwy. W takiej sytuacji trzeba przeszukać zarówno lewe, jak i prawe poddrzewo. Czas wyszukiwania jest zatem proporcjonalny do liczby niewyspecyfikowanych atrybutów.

Niech t oznacza liczbę niewyspecyfikowanych atrybutów w zapytaniu częściowym. Złożoność realizacji zapytania punktowego jest w k–d drzewie rzędu $O(\log n)$, podczas gdy złożoność realizacji zapytania częściowego jest rzędu $O(n^{((k-t)/k)})$. Złożoność operacji usuwania w dużym stopniu zależy od umiejscowienia usuwanego węzła w strukturze k–d drzewa i wynosi od $O(n^{((k-t/k)})$ do $O(\log n)$.

Podobnie jak w przypadku jednowymiarowych drzew binarnych, dostęp do danych zawartych w strukturze k–d drzewa jest najbardziej efektywny wówczas, gdy drzewo to jest w pełni zrównoważone. Nie istnieją jednak algorytmy umożliwiające utrzymanie w pełni zrównoważonego k–d drzewa. W celu uniknięcia wyrodnienia k–d drzewa należy co pewien czas wykonywać jego reorganizację, która pod względem złożoności jest równoważna sortowaniu i może być wykonana w czasie $O(n \log n)$.

Nie istnieje jak dotychczas wielowymiarowy odpowiednik AVL drzewa. Oprócz konieczności reorganizacji, do wad drzew należy również to, że ich efektywność zmniejsza się wraz z potęgą liczby k. Niezaprzeczalną zaletą k–d drzew jest natomiast dość duża efektywność realizacji zapytań wielowymiarowych w porównaniu z tradycyjnymi strukturami. Stosowanie k–d drzew jest sensowne wówczas, gdy liczba jego węzłów jest na tyle duża, że liczba poziomów przekracza liczbę k.

SPIS TREŚCI ROCZNIKA
INFORMATYKA 1990

ARTYKUŁY PROBLEMOWE

	nr	s.
BANACHOWSKI LECH, DEMINET JAROSŁAW, LAO J. MAREK – Propozycja nowego programu uniwersyteckich studiów informatycznych		
Cz. 1	2	13
Cz. 2	3	23
Cz. 3	4	21
BIAŁAS ANDRZEJ, BOJDAK ALEKSANDRA – Sieci MAP i TOP – przegląd standardów i przykład aplikacji	7	9
BLUEMKE ILONA – Architektury procesorów tablicowych tolerujących błędy	4	5
BRONOWSKA MARIA – System graficzny Ramtek RM – 9460	5	21
BUDZISZ HENRYK – Systemy ekspertowe w projektowaniu układów analogowych	11–12	27
CABAŃSKI JAN GRZEGORZ, LEWANDOWSKI MAREK, MARACH STANISŁAW, OCZKI ZBIGNIEW – ASTER – automatyczny sterownik transmisji szeregowych dla PC XT/AT	3	6
CYLKOWSKI ZBIGNIEW – Pomiar czasu obliczeń na IBM PC	9	11
CZEPIELEWSKI SŁAWOMIR, NOWAKOWSKI ZDZISŁAW – Środowisko programowe do projektowania pakietów elektronicznych systemów mikroprocesorowych	4	11
DAWIDOWSKI JERZY – Ręczne rejestraty danych	6	8
DEMBIŃSKI PIOTR – Protokoły komunikacyjne		
Cz. 1	5	13
Cz. 2 Model ISO	6	24
DWORAKOWSKI KRZYSZTOF, KLIMEK HENRYK, KORNIAK TADEUSZ, PRZEPIÓRA ELŻBIETA – Przechowywanie i przetwarzanie danych o nieregularnych strukturach	11–12	10
DYDEK ANDRZEJ – Porządkowanie zbiorów danych według polskiego alfabetu	1	6
FUGLEWICZ PIOTR – System CASE – problemy, techniki, rozwiązania		
Cz. 1	11–12	3
GONDZIO MAREK, PAWŁOWSKI MAREK, WYTRĘBOWICZ JACEK – Uruchamianie urządzeń mikroprogramowych		
Cz. 2	1	24
GÓRECKI ANDRZEJ – Historyczna szansa polskiej informatyki	9	5
GRABSKA EWA, HLINIAK GRAŻYNA, LEMBAS JACEK – Zastosowanie grafów kompozycyjnych do konstrukcji strukturalnego edytora graficznego	10	23
HOLT C. RICHARD (tłum. i oprac. **Janusz Zalewski**) – Turing – język lepszy od Pascala	3	1
HOŁYŃSKI MAREK (tłum. i oprac. **Janusz Zalewski**) – Sprzęg graficzny jako system ekspertowy	1	2
HUZAR ZBIGNIEW, KUŹNIARZ LUDWIK – Język Lotos		
Cz. 1. Specyfikacje typów danych	1	19
Cz. 2. Specyfikacje procesów	2	21
Cz. 3. Specyfikacje systemów	3	26
JURCZAK LUBOMIR, WARSKI WOJCIECH – System baz danych CX-DMOS		
Cz. 1	7	1
Cz. 2	8	24
KARDASIEWICZ JACEK – System MikroAPO. Podsystem projektowania technologii dla typoszeregów części	4	1
KŁOPOTEK MIECZYSŁAW, KOWALSKI ADAM, MACURA ROBERT, MICHALEWICZ MACIEJ, PACAN ANDRZEJ, SYROPIATKO ELŻBIETA, WIERZCHOŃ SŁAWOMIR – SYS86988 – System pozyskiwania i weryfikacji wiedzy medycznej		
Cz. 1	2	9
Cz. 2	3	15
KŁOPOTEK MIECZYSŁAW – Rozpoznawanie słów w mowie ciągłej	6	12
KŁOPOTEK MIECZYSŁAW – Medytor – mówiący edytor języka polskiego	11–12	15
KRÓLIKOWSKI ZBYSZKO – jednoczesne przetwarzanie wielu zapytań w relacyjnych bazach danych	6	1
KUBICKI WACŁAW, SUSKIEWICZ MARIAN – Możliwości stosowania światłowodów w krajowych systemach informatycznych	2	1
LANGE KATARZYNA – Poczta elektroniczna systemu Advanced Netware 286	11–12	32
ŁUBIŃSKA KATARZYNA – Rola Ady w wytwarzaniu oprogramowania	3	12
MALEC JAN – Maszyny lispowe		
Cz. 1	5	7
Cz. 2	6	19
MATERKA ANDRZEJ, GÓRNISIEWICZ WŁODZIMIERZ, KUKUŁA MARIUSZ – Mikrokomputerowe systemy przetwarzania obrazów	10	9
MOKRZYCKI WOJCIECH – Wprowadzenie do maszynowego postrzegania	10	1
MUSZYŃSKI MIŁOSZ – Fizyczne struktury systemów baz danych		
Cz. 1	8	1
Cz. 2. Techniki mieszania	9	17
Cz. 3. Wielowymiarowe struktury danych	11–12	23
NOWIŃSKI WIESŁAW L. Problemy współbieżnej rekonstrukcji obrazów	10	19
ORMIŃSKI MARIUSZ – Ethernet – od specyfikacji do realizacji praktycznej		
Cz. 1 – Specyfikacja	2	6
Cz. 2 – Realizacja praktyczna	3	19
ORWALDI PIOTR – Metody programowania pamięci EPROM	6	5

	nr	s.
PAPROCKI ANDRZEJ – Problemy implementacji języka Ada	1	9
POSTÓŁ MARIUSZ – Przemysłowe systemy czasu rzeczywistego		
Cz. 1. Rola i funkcje oprogramowania systemowego	8	7
Cz. 2. Struktura i funkcje biblioteki DIALOQUE	9	22
REKUĆ WITOLD – Odwzorowania w relacyjnej bazie danych obiektów o nieokreślonych a priori typach	8	15
RUDOLF R. WITOLD – System autorski JU–LEK	8	18
RUNKA HENRYK J. – Długi wektor w Turbo Pascalu 5.0	2	3
RUNKA HENRYK J. – Duża macierz w Turbo Pascalu 5.0	5	23
RYBIŃSKI HENRYK, SOBOLEWSKI PIOTR – GT–HOLMES – Graficzna baza danych	7	4
RYŻKO JAN, WRZESZCZ ZDZISŁAW – Optyczna pamięć dyskowa		
Cz. 1	8	10
SACH NGUYEN KIM – Wzmacnianie obrazów telewizyjnych metodami komputerowego przetwarzania obrazów	8	22
SACH NGUYEN KIM – Sprzętowe wzmacnianie obrazów cyfrowych w czasie rzeczywistym	7	20
SIENIAWSKI LESŁAW – Architektury zredukowane	5	1
SOBOL HENRYKA – Współbieżność sterowania bazą wideodanych i systemów przetwarzania obrazów	3	9
STOŻEK JAN – FidoNet – międzynarodowa amatorska sieć komputerowa	7	22
SZCZAWIŃSKA DOROTA, TRAWIŃSKI BOGDAN – Konwersacja danych między pakietami micro CDS/ISIS i dBase III Plus	7	13
WIERZBA ANDRZEJ MACIEJ– Edytor schematów elektronicznych	7	17
WRZESZCZ ZDZISŁAW – Język Lisp i elementy sztucznej inteligencji		
Cz. 1	9	1
Cz. 2 Konstruowanie funkcji i programu	11–12	18
ZALEWSKI JANUSZ – Okres istnienia oprogramowania		
Cz. 1	1	12
Cz. 2	2	18
Cz. 3	4	14
ZALEWSKI JANUSZ (oprac.) – Dwa kompilatory Moduli 2 IBM PC	2	26
ZIELIŃSKI KRZYSZTOF – Rozproszone przetwarzanie w grafice i transputery	9	7
ŻAKOWICZ TOMASZ – CDS/ISIS narzędzie do zarządzania bazami danych		
Cz. 1	5	16
Cz. 2	6	15
ŻYCHOWSKI PIOTR – Automatyczna standaryzacja słów tekstu w języku polskim	9	15

Z KRAJU

	nr	s.
Współczesne obszary zastosowań informatyki – **Piotr Fuglewicz**	1	27
Infosystem '89 – **Jarosław Deminet, Hanna Włodarska**	1	28
Profesor Antoni Kiliński (1907–1989)	3	22
Infosystem '90 – **Hanna Włodarska, Jarosław Deminet**	7	25
Ocena stanu polskiej informatyki (PTI)	9	26
III Walny Zjazd PTI – **Jarosław Deminet**	11–12	36

ZE ŚWIATA

	nr	s.
Sieci komputerowe najbardziej dynamiczną gałęzią gospodarki – oprac. **Jan Ryżko**	2	28
European CAMAC Association – oprac. **M. Kuc**	2	30
SDS – system wytwarzania oprogramowania w Moduli 2 – oprac. **J. Zal.**	3	29
Symboliczny program uruchomieniowy dla Ady – oprac. **Dorota Inkielman**		
Cz. 1	4	24
Cz. 2	5	27
Nowa generacja sieci lokalnych – oprac. **J. R.**	4	26
Tranzystory kwantowe – oprac. **J. Ryżko**	4	27
Trzy systemy programowania w Moduli 2 – oprac. **J. Zal.**	4	28
Oprogramowanie systemu sieciowego IBM – **Krzysztof Rzymkowski**	4	31
Ada dla mikrokomputerów MacIntosh – oprac. **M. Kuc**	6	28
Informatyka na wyższych uczelniach RFN – **WK**	6	30
Nowe systemy RISC – **JR**	6	31
Nieprawidłowe wykorzystanie sprzętu w RFN – **WK**	6	31
Współczesne pamięci masowe komputerów osobistych – **WK**	7	26
Rynek pamięci operacyjnych – **JR**	7	29
Drukarka laserowa firmy Hewlett-Packard – **JR**	7	30
Stowarzyszenie użytkowników informatyki w NRD – **WK**	7	30
Współpraca Siemensa z IBM – **JR**	7	30
Komputery do obliczeń naukowych – tłum. i oprac. **J. Ryżko**	8	28
Technologie układów scalonych w latach dziewięćdziesiątych – **JR**	8	29
Pierwsza dziesiątka komputerów osobistych – **JR**	8	30
Systemy 486 w Europie	8	30
Węgierski mikroprocesor	8	30

NOWE KSIĄŻKI

	nr	s.
Wydawnictwo Czasopism i Książek Technicznych NOT SIGMA	2	31
Nowe książki z dziedziny poligraficznego składu komputerowego	3	4
Książki z dziedziny sztucznej inteligencji rozprowadzane przez firmę J. Wiley	4	9
Wydawnictwa Naukowo-Techniczne	5	29
Wydawnictwa Naukowo-Techniczne	6	10
Wydawnictwa Naukowo-Techniczne	7	28
Wydawnictwa Naukowo-Techniczne	8	31
Wydawnictwa Naukowo-Techniczne	9	27
Wydawnictwa Naukowo-Techniczne	11–12	35

IMPREZY

	nr	s.
ASSOCIATION FOR COMPUTING MACHINERY 1991 COMPUTER SCIENCE CONFERENCE	5	29
EUROMICRO'90	6	7
Informatyka i ochrona środowiska	6	22

	nr	s.
Zintegrowane, inteligentne systemy informatyczne	7	3
CASE	7	30
Informatyka w Szkole VI	7	III okł.
SMAU'90 – **WK**	9	13
Grafika Komputerowa i Przetwarzanie Obrazów – **WM**	10	2
EUROGRAPHICS '91	11–12	14
Computer Networks'91	11–12	14
II Wiosenna Beskidzka Szkoła PTI	11–12	21

REFLEKSJE

Odpowiedzialność za jakość wyrobów – **WK**	5	III okł.
Bezpieczeństwo danych – **WK**	6	III okł.

zBITki

RESUME – Wacław Iszkowski	7	III okł.
ON WIRUS GOTO Gdziekolwiek – **Wacław Iszkowski**	8	III okł.
1990 PRINT „BASIC ma 26 lat" – **Wacław Iszkowski**	9	III okł.
IF Informatyk THEN Kto? ELSE Programista – **Wacław Iszkowski**	10	III okł.
LET DRENAŻ = POMOC – **Wacław Iszkowski**	11–12	III okł.

TERMINOLOGIA

Polskie znaki i klawiatury w normach międzynarodowych – **Janusz Zalewski**	1	III okł.
Atrybuty jakości oprogramowania – definicje TAEA – **Janusz Zalewski**	2	III okł.
Atrybuty jakości oprogramowania – definicje IEEE – **Janusz Zalewski**	3	III okł.
Atrybuty jakości oprogramowania – definicje podręcznikowe – **Janusz Zalewski**	4	III okł.

LISTY

Ulepszona wersja programu porządkującego według polskiego alfabetu – **Piotr Zienkiewicz**	7	24

/ unix

The Design and Implementation of the 4. 3BSD UNIX Operating System – **Jacek Surma**	9	28
Systemy otwarte i ich wrogowie, czyli co dalej z UNIX-em (1). Rozłam w UNIX-ie – **Jacek Surman**	9	29
Jak startuje UNIX – **Dariusz Kupiecki**	10	28
Systemy otwarte i ich wrogowie, czyli co dalej z UNIX-em (2). Główni konkurenci – **Jacek Surma**	10	29
Kosz na śmieci – **Władysław Majerski**	10	29
Co po czym i kiedy (O ewaluacji wejścia w shellu Bourne'a – **Jan Walasek**	11–12	38
Systemy otwarte i ich wrogowie, czyli co dalej z UNIX-em (3). Czy grozi nam standardowy UNIX? – **Jacek Surma**	11–12	40
Czy mógłbyś mnie zabić? (czyli o utracie kontaktu z rzeczywistością) – **Dariusz Kupiecki**	11–12	43
Argumenty, parametry, środowisko (w shellu Bourne'a) – **Jan Walasek**	11–12	44
Mimika pocztowa – **JS**	11–12	46
The Design of the Unix Operating System – **Jacek Surma**	11–12	46

Microma

ZAKŁAD SYSTEMÓW MIKROKOMPUTEROWYCH

MiCOMP

40–045 Katowice, ul. Astrów 7
telefon 513-086, telefaks 518-628,
teleks 315687

Systemy teletransmisji danych ICL/ODRA

Integracja systemów ICL i ODRA z mikrokomputerami standardu PC i systemami NOVELL, UNIX SCO

procesor sieci rozległej WAN
MiCOMP 8075
(emulacja ICL 7503)

most NOVELL – ICL/ODRA
NetCROSS-FXBM

program teletransmisji danych **MiCROS-FXBM** (PC, rozproszone bazy danych, wersja sieciowa – NOVELL), skaner MPXSCAN-8007,

adaptery, modemy, testery, emulatory terminali.

Instalacje w największych systemach na terenie całego kraju!

E0/11124/89

Cennik ogłoszeń w miesięczniku INFORMATYKA

● cała strona	600 000 zł
● 3/4 strony	540 000 zł
● 2/3 strony	480 000 zł
● 1/2 strony	360 000 zł
● 1/3 strony	240 000 zł
● 1/4 strony	180 000 zł
● 1/8 strony	120 000 zł
● poniżej 1/8 strony	2500 zł/cm²

Dopłaty:

● za dodatkowy kolor	+ 30%
● za I i IV stronę okładki	+ 100%
● za II i III stronę okładki	+ 50%

Rabaty:

● za powtórzenie 3–6 razy	– 10%
● za powtórzenie ponad 6 razy	– 20%
● za teksty o charakterze informacji techniczno-handlowej (artykuły)	– 50%

Powyższe ceny uwzględniają wszystkie koszty przygotowania ogłoszenia lub informacji techniczno-handlowej do druku.

W przypadku rezygnacji z ogłoszenia przed przekazaniem do druku, Zamawiający ponosi koszty w wysokości 25% wartości złożonego zamówienia. Przy rezygnacji po przekazaniu do druku, Zamawiający ponosi pełny jego koszt. Zmiana ustalonej przy zamówieniu treści ogłoszenia po przekazaniu do druku powoduje zwiększenie należności o 20%, lecz tylko za jedno ogłoszenie (koszt wykonania nowej kliszy).

Zamówienia na ogłoszenia przyjmują upoważnieni przez Zakład Wydawniczy INFORMATYKA akwizytorzy, którzy udzielają wszelkich wyjaśnień oraz dokonują szczegółowych uzgodnień technicznych i finansowych z Zamawiającym.

Zapewniamy szybką i terminową realizację ogłoszeń!

Nasz adres: INFORMATYKA, 01–552 Warszawa, Pl. Inwalidów 10, p. 104, tel. 39 14 34

Wskazówki dla Autorów

Nadsyłane artykuły nie mogą być publikowane lub przeznaczone do opublikowania w innych czasopismach.

Materiał oprócz tekstu zasadniczego powinien zawierać:

- *krótki życiorys zawodowy Autora i jego zdjęcie,*
- *wykaz literatury,*
- *tabele,*
- *materiał ilustracyjny (rysunki, zdjęcia czarno-białe, wydruki) dołączony do artykułu (nie wklejać materiału w tekst).*
- *podpisy pod ilustracje.*

Na osobnej stronie prosimy podać tytuł naukowy, imię i nazwisko, nazwę zakładu pracy, numer telefonu oraz informację, jaką drogą przekazać honorarium – kasa Wydawnictwa, poczta, bank.

Tekst artykułu prosimy dostarczyć w formie maszynopisu lub wydruku komputerowego, pisany jednostronnie, z podwójnym odstępem – bardzo ważne! – czyli 30 wierszy na stronie i 60 znaków w wierszu.

Wykaz literatury w porządku alfabetycznym.

Tabele – każda na osobnej stronie – powinny być numerowane i opatrzone tytułem oraz ściśle związane z tekstem (zaznaczone miejsce tabeli w tekście).

Rysunki – winy być czytelne (zaznaczyć ich miejsce w tekście), mogą być wykonane ołówkiem.

Zdjęcia – czarno-białe, kontrastowe, wykonane na błyszczącym papierze.

Wydruki – czytelne, kontrastowe, wykonane na białym papierze. Format – maksymalnie 18 cm w podstawie.

Podpisy pod ilustracje powinny być napisane na oddzielnej stronie, oprócz kolejnego numeru powinny zawierać tytuł (rysunku, zdjęcia, wydruku) i ew. legendę dotyczącą poszczególnych elementów.

Autor opublikowanego w INFORMATYCE artykułu otrzymuje honorarium i bezpłatny egzemplarz okazowy.

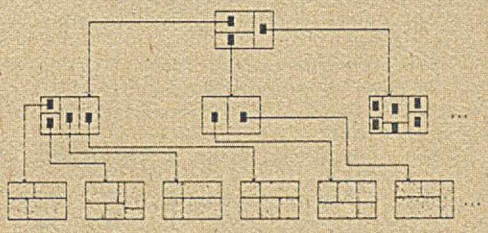
Materiałów nie zamówionych redakcja nie zwraca.

k-d-B drzewa

Nazwa *k*-d-B drzew [10] odzwierciedla ich pokrewieństwo z *k*-d drzewami i B drzewami. *k*-d-B drzewa są drzewami silnie rozgałęzionymi przeznaczonymi do zarządzania danymi w pamięci zewnętrznej. Analogicznie do B-drzew, *k*-d-B drzewa są zbiorem dwóch rodzajów węzłów o odmiennej strukturze: węzłów pośrednich oraz liści. Niech *s* oznacza stopień *k*-d-B drzewa. Węzeł pośredni *k*-d-B drzewa zawiera *s* par o postaci (*region, wskaźnik _węzła*), przy czym *region* jest to następujący zbiór par współrzędnych $/min_0, max_0, min_1, max_1, ...min_{k-1}, max_{k-1})$ w *k*–wymiarowej przestrzeni wszystkich rekordów pliku, a *wskaźnik _węzła* wskazuje na jedno z poddrzew tego węzła. Liść *k*-d-B drzewa zawiera pewną, w ogólności różną od *s*, liczbę par o postaci (*punkt, wskaźnik_rekordu*), przy czym punkt jest to zbiór współrzędnych $(x_0, x_1, ..., x_{k-1})$ w *k*-wymiarowej przestrzeni wszystkich rekordów pliku, a *wskaźnik_rekordu* jest wskaźnikiem do rekordu danych identyfikowanego przez wartości atrybutów $A_0, ..., A_{k-1}$ reprezentowane tutaj jako współrzędne punktu (to znaczy spełniające warunek $A_0 = x_0, A_1 = x_1, ..., A_{k-1} = x_{k-1}$).

W *k*-d-B drzewach nie występuje pojęcie dyskryminatora węzła. Zastosowana koncepcja jest podobna do koncepcji przyjętej w plikach kratowych. Każdy poziom *k*-d-B drzewa reprezentuje podział przestrzeni rekordów na *k*-wymiarowe prostopadłościany. Węzły pośrednie kolejnych poziomów są analogami wielopoziomowego katalogu pliku kratowego.

W *k*-d-B drzewach obowiązuje zasada, że wszystkie regiony są rozłączne, a ich suma jest także regionem. Dla każdej pary (*region, wskaźnik_węzła*) *region* stanowi sumę wszystkich regionów w węźle wskazywanym przez *wskaźnik_węzła*. Jeżeli węzeł wskazywany przez *wskaźnik_węzła* jest liściem, to wszystkie punkty będące elementami tego liścia są zawarte w regionie *region*. Powyższe zasady umożliwiają realizację efektywnych algorytmów wyszukiwania, wstawiania i usuwania rekordów w *k*-d-B drzewie. Problem przepełnienia węzła rozwiązuje się podobnie jak w pliku kratowym i w B-drzewie. Przepełnienie liścia wiąże się z utworzeniem nowego liścia i przemieszczeniem do niego części rekordów z liścia przepełnionego. Rozdzielenie liścia i utworzenie dwóch liści wiąże się z koniecznością zastąpienia pary (*region, wskaźnik_liścia*) w bezpośrednio nadrzędnym węźle pośrednim dwiema parami: $/region_1$, *wskaźnik_liścia_starego*) oraz ($region_2$, *wskaźnik_liścia_nowego*), przy czym suma regionów $region_1$ i $region_2$ stanowi region *region*.



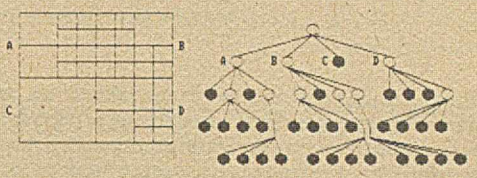
Rys. 5

Podział regionu, podobnie jak w plikach kratowych, następuje wskutek dodania dodatkowego przedziału na wybranej osi przestrzeni rekordów. Zastąpienie jednej pary (*region, wskaźnik_węzła*) dwiema parami może spowodować przepełnienie węzła pośredniego. W takiej sytuacji dokonuje się rozdziału węzła pośredniego. W szczególnym przypadku wstawienie jednego rekordu do bazy danych może doprowadzić do zmiany wysokości *k*-d-B drzewa. Podobnie realizuje się usuwanie z *k*-d-B drzewa przy założeniu, że wypełnienie węzła nie powinno być mniejsze niż 1/2. Przykładowe *k*-d-B drzewo dla $k=2$ przedstawiono na rys. 5.

Quad drzewa

Quad drzewa umożliwiają przechowywanie i efektywne operowanie danymi dwuwymiarowymi. Struktura ta została zaproponowana przede wszystkim jako struktura do zastosowań graficznych, umożliwiających przechowywanie opisów dwuwymiarowych graficznych obiektów w hierarchicznie zdekomponowanej strukturalnej postaci. W dużym uproszczeniu quad drzewa można rozpatrywać jako szczególny przypadek *k*-d-B drzewa o dwóch wymiarach i stopniu równym cztery.

Nazwa quad drzewa wiąże się z podstawową zasadą dekompozycji obiektu reprezentowanego w jego strukturze, polegającą na podziale opisywanego obiektu na cztery części. Korzeń drzewa reprezentuje cały opisywany obiekt; cztery następniki korzenia reprezentują cztery części tego obiektu, a węzły będące następnikami kolejnych poziomów korzenia reprezentują coraz to drobniejsze części opisywanego obiektu (rys. 6).



Rys. 6

Quad drzewa można stosować do przechowywania obrazów w pamięci komputera, co ma poważne zalety w porównaniu z przechowywaniem obrazów w postaci mapy bitowej. Zapisywanie obrazów w postaci mapy bitowej jest bardzo pamięciochłonne, aby więc zmniejszyć zajmowaną pamięć, stosuje się kompresję, korzystając z faktu, że wiele piksli obrazu, zwłaszcza sąsiednich, ma tę samą wartość atrybutów wyświetlania. Ciągi bajtów hoduje się jako pojedyncze wzorce bajtów, poprzedzone bajtami określającymi liczbę wystąpień. Takie techniki mają jednak dwie zasadnicze wady:

- kompresja uwzględnia zależności między pikslami tylko w jednym wymiarze, podczas gdy w rzeczywistości zależności te są dwuwymiarowe,
- na obrazie przechowywanym w pamięci w postaci skompresowanej nie można wykonywać żadnych operacji.

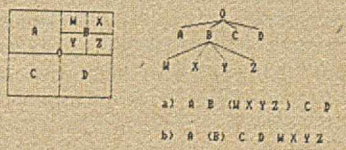
Quad drzewa umożliwiają przechowywanie obrazów w pamięci strukturalnej, zapewniając jednocześnie kompresję obrazu z wykorzystaniem dwuwymiarowych zależności między pikslami.

Każdy węzeł quad drzewa może być liściem lub też może mieć jeden, dwa, trzy lub cztery następniki. Liście quad drzewa zawierają informacje o pikslach obrazu reprezentowanego w maksymalnej rozdzielczości. Charakterystyczną cechą quad drzew jest to, że na podstawie węzłów określonego poziomu można otrzymać reprezentację obrazu o dowolnie wybranej rozdzielczości. Jest to możliwe oczywiście tylko wówczas, gdy w węzłach pośrednich jest przechowywana informacja o średnich wartościach danych zawartych w liściach poddrzew tych węzłów. Możliwość odtworzenia obrazu ze zmniejszoną rozdzielczością ma zastosowanie w animacji komputerowej, w której przy obrazach statycznych wykorzystuje się maksymalną rozdzielczość, a podczas generowania obrazów dynamicznych rozdzielczość może zostać zmniejszona w celu uzyskania większej szybkości odtwarzania obrazów.

Quad drzewo jest zazwyczaj kodowane w pamięci bez użycia wskaźników; są one bowiem na ogół dłuższe niż pola odpowiadające atrybutom wyświetlania pojedynczych piksli, przez co pamięć zajęta przez wskaźniki kilkukrotnie przekraczałaby pamięć zajętą przez liście quad drzewa. Do kodowania quad drzewa bez wskaźników stosuje się zazwyczaj dwie metody:

- pionową (ang. *depth first*),
- poziomą (ang. *breadth first*).

Metoda pionowa polega na umieszczeniu węzłów quad drzewa w pamięci w takiej kolejności, aby zawsze najpierw występowały węzły leżące niżej w hierarchii quad drzewa, a dopiero potem węzły równorzędnego poziomu. Metoda pozioma zakłada natomiast porządek odpowiadający pozycjom węzłów w hierarchii quad drzewa (rys. 7). Należy zaznaczyć, że w quad drzewie trzeba zapewnić możliwości rozróżnienia, które węzły są liśćmi, gdyż mogą one wystąpić na dowolnym poziomie.



Rys. 7

Istnieją efektywne algorytmy, umożliwiające wykonywanie różnych operacji na obrazach przechowywanych w strukturach quad drzew, na przykład obrót, odwrócenie obrazu, łączenie dwóch obrazów, wypełnianie obszarów [1, 8, 9, 13, 14].

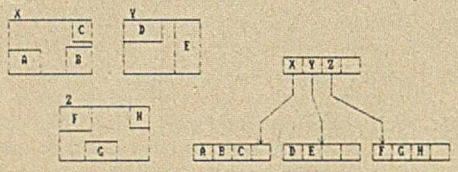
R drzewa

R drzewa zostały zaproponowane jako struktura danych do zastosowań w obiektowych bazach danych [3, 4, 11]. Obiektem można nazwać każdą umowną jednostkę informacji, dającą się odróżnić od innych jednostek. Opisem obiektu jest zbiór danych umożliwiający dokonanie tego odróżnienia. W obiektowych bazach danych dopuszcza się istnienie wielu typów obiektów, z których każdy jest reprezentowany przez odmienną stukturę danych.

W organizacji dostępu z użyciem R drzew zakłada się, że wszystkie obiekty zajmują miejsce w pewnej k-wymiarowej przestrzeni. Wyszukiwanie rekordów odbywa się przez określenie współrzędnych obiektu w tej przestrzeni. Stanowi to pewne uproszczenie, ponieważ w ogólnym przypadku w obiektowych bazach nie wszystkie obiekty można rozpatrywać w kategoriach ich umieszczenia w jednej przestrzeni. R drzewa są zatem szczególnie predystynowane do zarządzania obiektami graficznymi, dwu- lub trójwymiarowymi.

R drzewa składają się z węzłów pośrednich oraz liści. Każdy liść R drzewa stanowi parę (*RECT, oid*), w której *oid* jest identyfikatorem obiektu rozumianym jako wskaźnik do opisu obiektu, natomiast *RECT* jest zbiorem współrzędnych określających minimalny k-wymiarowy prostopadłościan zawierający ten obiekt. Węzeł pośredni R drzewa jest zbiorem par o postaci (*RECT, p*), w których *p* jest wskaźnikiem do jednego z następników tego węzła, a *RECT* jest zbiorem współrzędnych określających minimalny k-wymiarowy prostopadłościan zawierający wszystkie obiekty wskazywane przez następniki tego węzła (rys. 8).

Struktura R drzewa jest podobna do struktury k-d-B drzewa. Podstawową różnicą jest brak elementów atomowych w liściach drzewa; inny jest również sposób interpretacji k-wymiarowych prostopadłościanów zawartych w węzłach drzewa. W k-d-B drzewach cała reprezentowana przestrzeń była objęta przez k-wymiarowe prostopadłościany zdefiniowane w węzłach drzewa, w R drzewach natomiast jest objęta jedynie ta przestrzeń, która nie jest pusta i zawiera obiekty.



Rys. 8

Przez **pokrycie** R drzewa rozumie się sumę objętości wszystkich k-wymiarowych prostopadłościanów zawartych w liściach. Z pokryciem wiąże się problem martwej przestrzeni, przy dużym pokryciu jest bowiem możliwe, że przestrzeń określona przez element *RECT* nie jest całkowicie wypełniona przez zawarty w niej obiekt. Punkty zawarte w granicach elementu *RECT* i nie wchodzące w skład żadnego z obiektów są nazywane **punktami martwej przestrzeni**. Punkty takie są niedostępne dla algorytmu wstawiania nowych obiektów do R drzewa. Kryterium oceny algorytmu wstawiania jest więc minimalne pokrycie.

Pojęcie nakładania wiąże się z problemem towarzyszącym przypadkowemu wstawianiu nowych elementów do R drzewa. Przy wstawianiu może powstać sytuacja, w której obszary określone przez dwa elementy *RECT* w dwóch różnych liściach będą się nakładać. Efektem jest istnienie więcej niż jednej drogi wiodącej od korzenia do konkretnego obszaru, co powoduje zmniejszenie efektywności wyszukiwania. W celu uniknięcia problemu pokrycia wprowadzono modyfikację R drzewa, zwaną R+ drzewem. R+ drzewa umożliwiają rozdzielenie jednego z pokrywających się prostopadłościanów na mniejsze elementy przechowywane w oddzielnych liściach. Podział ten jest wykonywany w taki sposób, aby suma elementów pokrywała cały obiekt, lez żaden z nich nie nakładał się na inny prostopadłościan. Brak nakładania został więc w R+ drzewach osiągnięty kosztem zwiększonego zapotrzebowania na pamięć.

* * *

W artykule omówiono dwa zasadnicze typy organizacji jednowymiarowych struktur danych w bazach danych: drzewa, określane także mianem technik indeksowania, oraz techniki mieszania. Wybór jednej z nich do implementacji konkretnego systemu bazy danych zależy od odpowiedzi na dwa pytania:

- czy ważniejszy jest średni czas dostępu do danych, czy też czas dostępu w najgorszym przypadku,
- jakie typy zapytań najczęściej będą przedstawiane w realizowanej bazie danych.

Dla zapytań punktowych technika mieszania jest bardziej efektywna – średni czas dostępu do danych jest zdecydowanie mniejszy. Czas odpowiedzi na zapytanie punktowe z użyciem indeksu jest rzędu $O(\log n)$, a z wykorzystaniem technik mieszania jest rzędu $O(1)$. Techniki mieszania charakteryzują się jednak bardzo długim czasem dostępu, w najgorszym przypadku wynoszącym $O(n)$; czas dostępu w najgorszym przypadku dla technik indeksowania zazwyczaj nie różni się od średniego czasu dostępu (lub różni w niewielkim stopniu).

Dla zapytań z zakresem technika indeksowania jest zdecydowanie bardziej efektywna. Wystarczy znaleźć rekord z wartością atrybutu odpowiadającą granicy wyspecyfikowanego zakresu, a następnie korzystając z sekwencyjnej części struktury indeksu pobierać kolejne rekordy aż do napotkania rekordu z wartością atrybutu odpowiadającą drugiej granicy zakresu. Funkcja mieszająca jest bardzo nieefektywna dla zapytań z zakresem, nie istnieje bowiem dla niej pojęcie następnego rekordu. Są specjalne odmiany funkcji mieszających zachowujących porządek, jednak znalezienie funkcji mieszającej zachowującej porządek, równomierność i losowość rozkładu jest w większości przypadków zadaniem bardzo trudnym. Z tego powodu w systemach bez danych stosuje się raczej techniki indeksowania.

Omówione wielowymiarowe struktury danych stanowią jedynie wstępny zarys obszernej i niezwykle szybko rozwijającej się dziedziny. Wszystkie przedstawione wielowymiarowe struktury danych można zaliczyć do wielowymiarowych technik indeksowania, chociaż istnieją też wielowymiarowe modyfikacje technik mieszania. Zastosowanie wielu struktur jest często w poważny sposób ograniczone, na przykład, pliki kratowe stosuje się jedynie do zarządzania zbiorami rekordów o małej liczbie atrybutów ($k < 10$) o wartościach z obszernych, liniowo uporządkowanych dziedzin. Możliwości wielu struktur nie zostały jeszcze do końca przeanalizowane, dotyczy to zwłaszcza struktur proponowanych do zastosowań w obiektowych bazach danych.

LITERATURA

[1] Anedda C., Felician L.: P-compressed Quadtrees for Imagé Storing. The Computer Journal, V. 31, No. 4, pp. 353–357, 1988

[2] Bentley J. L.: Multidimensional Binary Search Trees Used for Associative Searching. Communications of the ACM. Vol. 18, No. 9, p. 509–517, 1975

[3] Faloutsos C., Sellis T. N.: Analysis of Object Oriented Spatial Access Methods. Proc. of ACM SIGMOD, pp. 426–439, 1987

[4] Guttman A.: R-tries: a Dynamic Index Structure for Spatial Searching. Proc. of ACM SIGMOD, pp. 47–57, June 1984

[5] Johnson J. S., Webster D. B.: Updating an Inverted Index – a Performance Comparison of Two Techniques. The Computer Journal, Vol. 25, No. 2, pp. 169–175, 1982

[6] Nicklas B. M., Schlageter G.: Index Structuring in Inverted Data Bases by Tries. The Computer Journal, Vol. 20, No. 4, pp. 321–324, 1977

[7] Nievergelt J., Hinterberger H., Sevcik K. C.: The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Transactions on Database Systems. Vol. 9, No. 1, pp. 38–71, March 1984

[8] Oliver M. A., Wiseman N. E.: Operations on Quadtree Encoded Images. The Computer Journal, Vol. 26, No. 1, pp. 83–92, 1983

[9] Oliver M. A., Wiseman N. E.: Operations on Quadtree Leaves and Related Image Areas. The Computer Journal. Vol. 26, No. 1, pp. 375–380, 1983

[10] Robinson J. T.: The k-d-B Tree – A Search Structure for Large Multidimensional Dynamic Indexes. Proc. of ACM SIGMOD, pp. 10–18, April 1981

[11] Roussopoulos N., Faloutsos C., Sellis T.: An Efficient Pictorial Database System for PSQL. IEEE Trans. on Soft. Eng., Vol. SE-14, No. 5, pp. 639–650, 1988

[12] Salzberg B.: Grid File Concurrency. Inform. Systems. Vol. 11, No. 3, pp. 235–244, 1986

[13] Williams R.: The Goblin Quadtree. The Computer Journal, Vol. 31, No. 4, pp. 358–363, 1988

[14] Woodwark J. R.: The Explicit Quad Tree as a Structure for Computer Graphics. The Computer Journal. Vol. 25, No. 2, pp. 235–238, 1982.

Bardzo prosimy autorów przysyłających teksty do redakcji o podawanie danych umożliwiających bezpośredni kontakt w godzinach pracy (numeru telefonu), adresu domowego i ewentualnie numeru konta bankowego.

HENRYK BUDZISZ
Zakład Elektroniki
Wyższa Szkoła Inżynierska
Koszalin

Systemy ekspertowe w projektowaniu układów analogowych

W technologii bipolarnej układy analogowe wytwarza się w długich seriach jako podzespoły o uniwersalnych zastosowaniach – nowe opracowania układów scalonych pojawiają się dość rzadko. Wraz z powstaniem możliwości wykonywania układów analogowych w technologii MOS o dużym stopniu scalenia (często wspólnie z układami cyfrowymi na jednym podłożu), stało się konieczne indywidualne projektowanie tych układów, na podstawie ściśle określonych specyfikacji, narzuconych przez funkcję, jaką układ ma spełniać. Wynika ona głównie z potrzeby ograniczenia poboru mocy, powierzchni zajmowanej przez układ oraz braku zewnętrznych elementów współpracujących z układem.

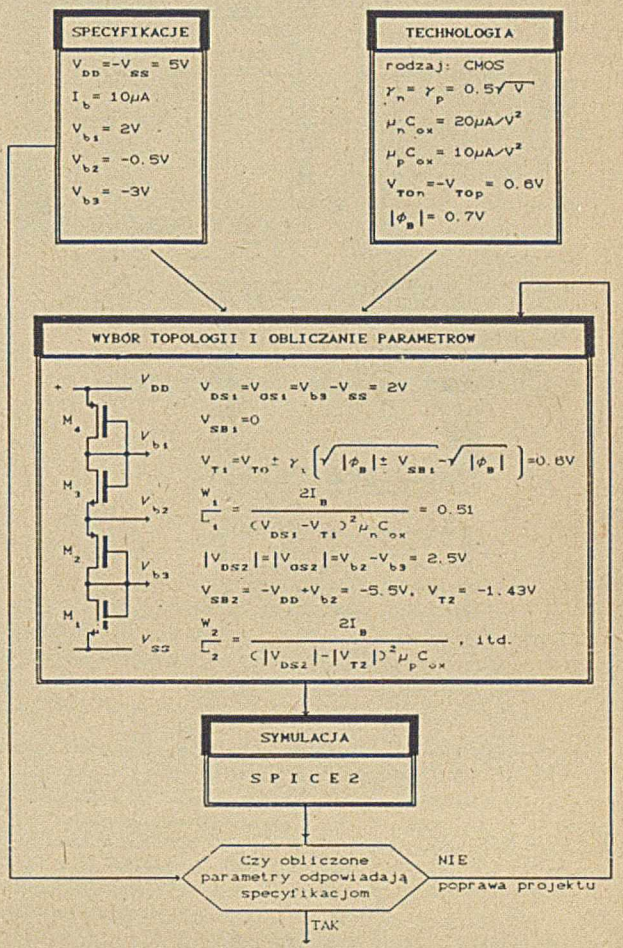
Projektowanie scalonych układów analogowych jest obecnie wspomagane różnymi programami narzędziowymi, jak edytory graficzne, symulatory lub programy weryfikujące projekt. Mimo takich ułatwień, procesy projektowania topologii układu i jego optymalizacji są długotrwałe i wymagają dużego doświadczenia w projektowaniu. Ponadto opanowanie umiejętności posługiwania się narzędziami wspomagającymi projektowanie wymaga często znacznego wysiłku ze strony projektanta. Rozwiązania tych problemów oczekuje się od powszechnego zastosowania systemów ekspertowych, które w odróżnieniu od narzędzi wspomagających projektowanie muszą odgrywać aktywną rolę w procesie projektowania. Prace o charakterze rozpoznawczym i prototypowe implementacje są realizowane od 1980 roku [14, 15]. Pierwsze użyteczne systemy ukazały się w latach 1986–1987, m.in. IDAC (Centre Suisse d'Electronique et de Microtechnique, Neuchatel, Szwajcaria). OASYS (Carnagie Mellon University, Pittsburgh, USA), OPASYN (University of California, Berkely, USA).

Schemat procesu projektowania „ręcznego" przedstawiono na rys. 1. W pierwszym cyklu projektant, na podstawie specyfikacji parametrów układu, określa jego topologię i parametry elementów, korzystając z uproszczonych zależności projektowych. Następnym krokiem jest symulacja układu, np. przy użyciu programu SPICE [13]. Ponieważ symulator używa znacznie dokładniejszych modeli elementów, wyniki symulacji zwykle różnią się od zadanych specyfikacji. Po zmianie parametrów elementów lub korekcie schematu, proces symulacji i porównywania powtarza się. Liczba i wielkość zmian w projekcie oraz liczba iteracji zależą od doświadczenia projektanta.

W dotychczas opracowanych lub sygnalizowanych systemach projektowania opartych na bazie wiedzy, można wyróżnić trzy podejścia:
- wybór gotowych topologii z biblioteki (np. IDAC [5], OPASYN [4]).
- projektowanie oparte na hierarchicznym podziale układu (np. OASYS [11], BLADES [7]),
- poszukiwanie nowych struktur układów [2, 12, 15].

W dwóch pierwszych metodologiach ważnym elementem systemu jest biblioteka schematów, wraz z towarzyszącymi im zależnościami projektowymi lub uproszczonymi modelami. Biblioteka ta jest częścią wiedzy o projektowaniu. Zadaniem komputera nie jest tworzenie nowych topologii, ale wybór właściwej dla danych specyfikacji oraz wyznaczenie wymiarów tranzystorów.

W pracy [21] przedstawiono system ekspertowy do weryfikowania formalnej poprawności i korekty opisu schematów. System współpracuje z innym systemem o nazwie AUTORED [20], który służy do wczytywania schematów do bazy danych za pomocą kamery przyłączonej do minikomputera VAX 11/780. Tak połączone systemy umożliwiają szybkie tworzenie dużych bibliotek układów.

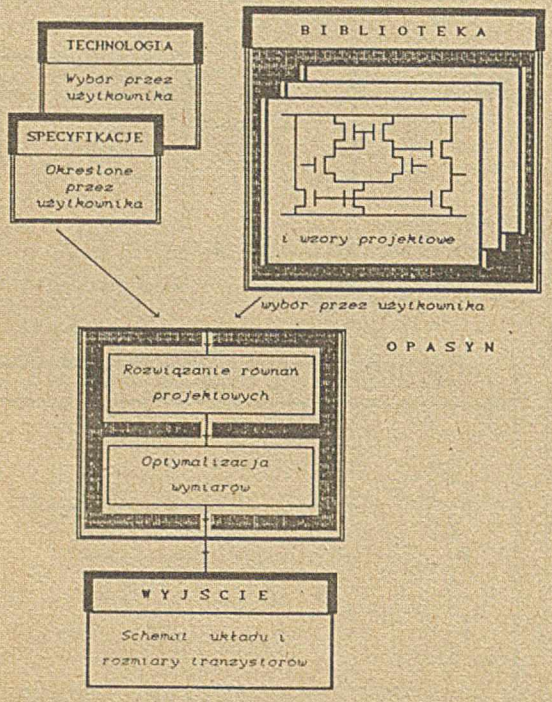


Rys. 1. Schemat projektowania „ręcznego"

Projektowanie przy ustalonej topologii

Jeżeli topologia układu została określona (wybrana z biblioteki lub podana w postaci symbolicznej), to pozostaje problem wyznaczania takich parametrów elementów, aby układ spełniał zadane specyfikacje. Matematycznie jest to problem wielowymiarowej optymalizacji z ograniczeniami; tak też potraktowano go w systemie OPASYN [4]. Biblioteka systemu zawiera kilkadziesiąt struktur topologicznych układów MOS, takich jak wzmacniacze transkonduktancyjne, wzmacniacze operacyjne, oscylatory kwarcowe. Każdemu układowi towarzyszą równania projektowe (rys. 2). Zadaniem systemu jest obsługa biblioteki i organizacja procesu optymalizacji przy użyciu procedur numerycznych. Jeżeli niektóre specyfikacje zostają osiągnięte z niezadowalającą dokładnością, to można wprowadzić współczynniki wagowe dla poszczególnych specyfikacji.

Zagadnienia optymalizacji należą do problemów złożonych, rys. 2, a obliczenia są zwykle czasochłonne (patrz tabela). Dlatego w innych rozwiązaniach stosuje się raczej podejście heurystyczne niż czysto algorytmiczne. Przykładem zastosowania takiej metodologii projektowania jest system IDAC [5]. Organizacja biblioteki jest podobna jak w poprzednim przykładzie. System nie ma możliwości rozwiązywania równań opisujących właściwości układów. Równania te muszą być wcześniej rozwiązane analitycznie, a do biblioteki wprowadza się, dla każdego układu, gotowe formuły projektowe. Dla wybranego przez użytkownika układu są obliczane jego parametry z towarzyszących mu zależności na podstawie zadanych specyfikacji i wartości domyślnych. Jeżeli obliczone wartości nie spełniają oczekiwań, to określa się nowe specyfikacje i powtarza się obliczenia (rys. 3). Ten typ iteracji, zwłaszcza w przypadku wymagań o sprzecznych oddziaływaniach nie zawsze kończy się sukcesem. Zaletą takiego podejścia jest duża szybkość działania (tab.), a wadą uciążliwe przygotowanie zależności projektowych i ograniczone możliwości wyboru. Jeżeli zadane specyfikacje nie zostaną osiągnięte, to projekt nie będzie wykonany, a system sygnalizuje przyczynę niepowodzenia.



Rys. 2. Schemat działania systemu OPASYN

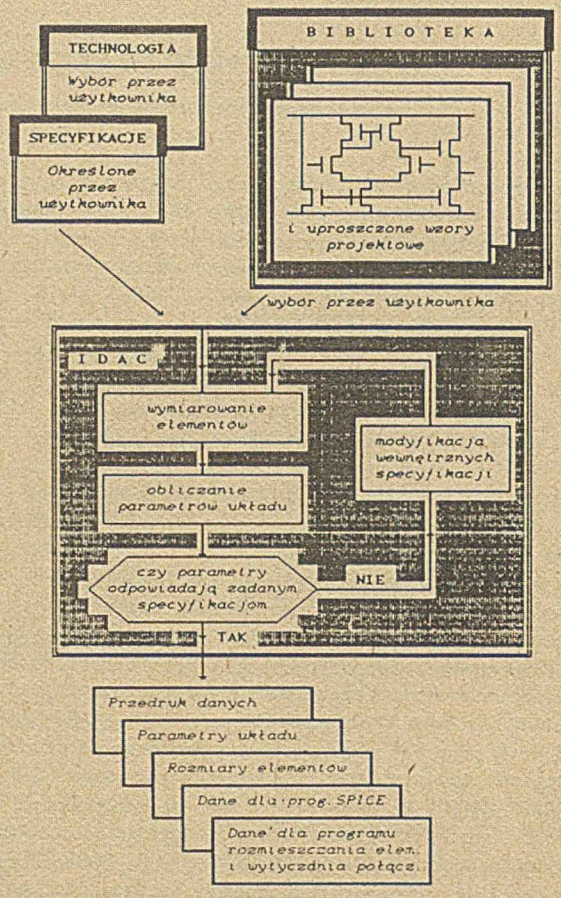
Porównanie średniego czasu projektowania wzmacniacza operacyjnego (komputer VAX-8800) [4] oraz sposobu rozwiązywania równań projektowych dla różnych systemów ekspertowych

System	Czas [sek]	Sposób rozwiązywania równań
IDAC	<1	Dla każdego schematu równania muszą być rozwiązane analitycznie. Schematom towarzyszą gotowe formuły projektowe. Iteracyjne poprawianie parametrów elementów.
OPASYN	70–280	Numeryczne rozwiązywanie równań. Wyznaczanie parametrów elementów przeprowadza się metodami optymalizacji (ekstremum funkcji wielu zmiennych).
OASYS	≃1	Hierarchiczne rozwiązywanie równań projektowych. Liczba równań dla każdego bloku jest niewielka. Parametry elementów poprawia się iteracyjnie

Systemy operujące ustalonymi topologiami mają dosyć ograniczone możliwości wyboru: zmiennymi są tam tylko rozmiary tranzystorów. Pewne ulepszenie wprowadzono w systemie CAMP [8]. Biblioteka zawiera również układy o ustalonej topologii, ale system może te topologie modyfikować przez wymianę bloków funkcjonalnych (ang. *self-reconstructing technique*).

W pracy [16] przedstawiono prototypowy system, który asystuje w procesie projektowania układu o ustalonej topologii. Jest to przykład systemu doradczego, który podaje, na przykład, co należy zrobić, aby zwiększyć pole wzmocnienia.

Zastosowanie metod propagacji ograniczeń [18] do syntezy układów pozwala wyznaczać wartości elementów dla ustalonej topologii i zadanych specyfikacji (system SYN) [17]. Poważnym ograniczeniem jest tu konieczność użycia metod przetwarzania symbolicznego wyrażeń algebraicznych oraz problemy z rozwiązywaniem sprzeczności (ang. *conflict resolution*). Pomyślne rezultaty osiągnięto dla niewielkich układów.



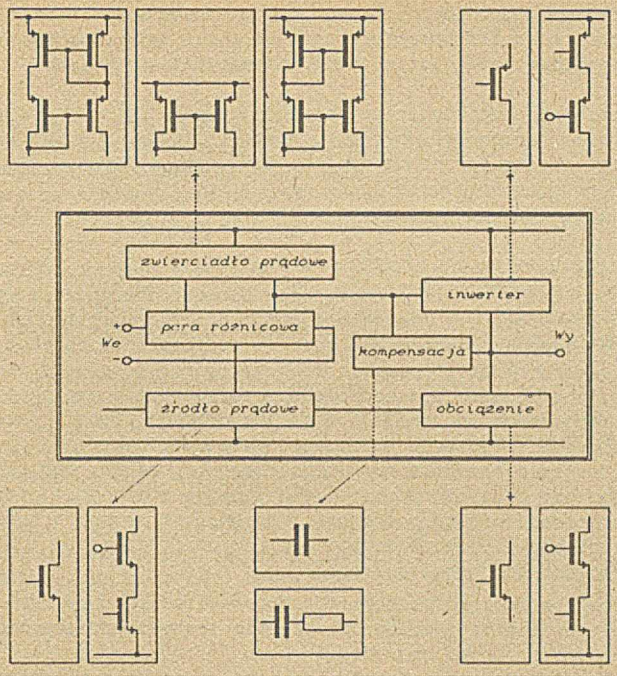
Rys. 3. Schemat działania systemu IDAC

Projektowanie hierarchiczne

Metodologia projektowania oparta na hierarchicznym podziale układu jest bardziej elastyczna. W pierwszym etapie projektowania wybiera się architekturę układu, określającą połączenia między poszczególnymi blokami (rys. 4).

Następnie specyfikacje parametrów całego układu dekomponuje się na specyfikacje poszczególnych bloków funkcjonalnych, a te z kolei – na specyfikacje mniejszych bloków, aż do osiągnięcia opisu pojedynczych tranzystorów. Na każdym poziomie hierarchii następuje wybór topologii poszczególnych bloków funkcjonalnych (rys. 4).

Typowym przykładem takiego podejścia jest system OASYS [11]. Na opis modułu składa się jego schemat (zbudowany z modułów niższego poziomu), równania projektowe oraz plan, według którego specyfikacje wejściowe są przekształcane na specyfikacje dla modułów niższego poziomu (rys. 5). Jeżeli na jakimś etapie parametry modułu lub elementu nie mogą być osiągnięte, to jest ustalana przyczyna niepowodzenia. W zależności od tej diagnozy następuje modyfikacja planu, równań projektowych lub topologii. Jeżeli mimo wszelkich modyfikacji, układ nie spełnia zadanych specyfikacji, to projekt nie zostanie wykonany. Poważną słabością systemów opartych na planowaniu akcji są trudności w ustaleniu przyczyn niewykonania projektu.



Rys. 4. Przykład architektury wzmacniacza

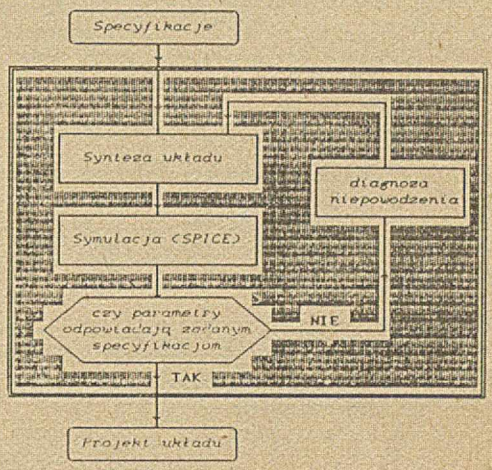
Resler [14] zaproponował hierarchiczną syntezę wzmacniacza operacyjnego, opartą na gramatyce bezkontekstowej. Program zawiera bibliotekę modułów, z których buduje się wzmacniacze według następujących reguł, podanych tu w notacji BNF:

⟨ wzmacniacz operacyjny ⟩:: = ⟨ pierwszy stopień ⟩ ⟨ drugi stopień ⟩ ⟨ trzeci stopień ⟩
⟨ pierwszy stopień ⟩:: = ⟨ prosta para różnicowa i ⟩ ⟨ obciążenie ⟩ |
⟨ para różnicowa Darlingtona i ⟩ ⟨ obciążenie ⟩ |
⟨ stopień różnicowy super beta z obciążeniem ⟩ |
⟨ stopień różnicowy z redukcją prądów i obciążeniem ⟩
⟨ obciążenie ⟩:: = ⟨ obciążenie rezystywne ⟩ | ⟨ proste źródło prądowe ⟩ | ⟨ złożone źródło prądowe ⟩
⟨ drugi stopień ⟩:: = ⟨ prosty układ ze wspólnym emiterem ⟩ |
⟨ układ Darlingtona ze wspólnym emiterem ⟩
⟨ trzeci stopień ⟩:: = ⟨ standardowy układ przeciwsobny ⟩
⟨ ulepszony układ przeciwsobny ⟩

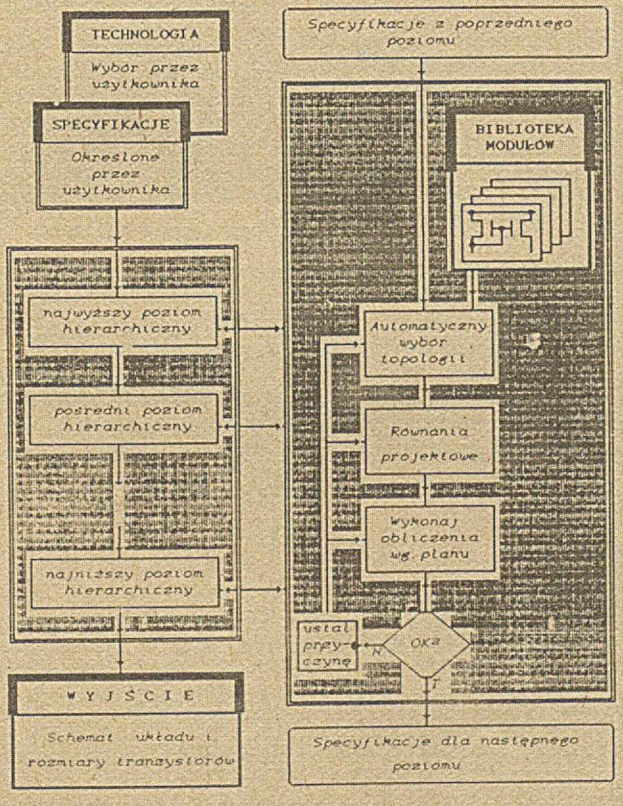
Ze względu na brak rekurencji gramatyka ta definiuje skończony zbiór wzmacniaczy (32 egzemplarze). W pierwszej wersji projektu, jeżeli moduł może być zrealizowany na kilka sposobów, wybiera się rozwiązanie najprostsze i wyznacza parametry wzmacniacza. Jeżeli nie spełniają one założonych specyfikacji, to zmienia się realizację tych modułów, które mogą poprawić osiągnięte rezultaty.

W systemie BLADES [7] do obsługi hierarchicznego procesu projektowania zastosowano technikę systemów rozproszonych. Każdemu modułowi towarzyszy podsystem ekspertowy, a całością zarządza system nadzorujący (ang. *design manager*).

W pracach [6, 22] zaproponowano strukturę systemu z weryfikacją, do której jest stosowany symulator SPICE. Wyniki symulacji służą do korekty parametrów lub topologii. Procesem tym steruje pomocniczy system diagnostyczny (rys. 6).



Rys. 6. Schemat blokowy systemu z weryfikacją



Rys. 5. Schemat działania systemu OASYS

Przewaga hierarchicznego podejścia do projektowania układów jest wyraźna. Zapewnia większą elastyczność projektowania i aktualizacji bazy wiedzy. Pozwala zastąpić jedno duże zadanie projektowe wieloma mniejszymi zadaniami. Wadą natomiast są ograniczenia w stosowaniu rozwiązań niekonwencjonalnych, „przeskakujących" strukturę hierarchiczną [11] – tzw. triki projektowe, często stosowane przez projektantów w celu „wyśrubowania" parametrów. Ponadto, pojedynczy element może pełnić w układzie kilka różnych funkcji, które bierze się pod uwagę na różnych szczeblach podziału hierarchicznego.

Ważnym problemem w każdym systemie przeznaczonym do projektowania układów jest strategia rozwiązywania równań projektowych. Równania te są często nieliniowe i tworzą układy silnie podkreślone (układy zawierające więcej zmiennych niż równań). Do określenia wartości niektórych zmiennych, używa się związków o charakterze heurystycznym, wynikających z doświadczenia ekspertów w tej dziedzinie. Strategia rozwiązywania równań w dużym stopniu decyduje o sprawności systemu (tab.). Związków między specyfikacjami a wartościami elementów nie należy mylić z równaniami stosowanymi w analizie, gdzie zmiennymi są prądy i napięcia, a wartości elementów są znane.

Synteza nowych struktur topologicznych

Oprócz prac nad systemami wykorzystującymi topologie zapisane w bazie wiedzy, prowadzi się próby syntezy nowych struktur. Jedną z pierwszych [15] była próba syntezy oparta na opisie funkcjonalnym elementów i układów oraz wnioskowaniu zstępującym. Dla przykładu syntezy generatora przebiegu jednostajnie rosnącego, cel został sformułowany następująco: zbudować układ, którego napięcie wyjściowe ma pochodną o stałej dodatniej wartości. Cel ten uruchamia reguły wnioskowania:

- pochodną napięcia węzłowego można sterować, jeżeli weźmie się pojemność, steruje się prądem płynącym przez tę pojemność, a napięcie na pojemności będzie tym napięciem węzłowym;
- jeżeli pochodna napięcia gałęziowego na pojemności ma być stała, to prąd płynący przez tę pojemność musi mieć stałą wartość;
- jeżeli prąd płynący przez element ma mieć stałą wartość, to do elementu należy przyłączyć źródło prądowe o stałej wydajności;
- jeżeli napięcie na elemencie ma być napięciem węzłowym, to jeden z zacisków elementu należy przyłączyć do masy, a drugi do tego węzła.

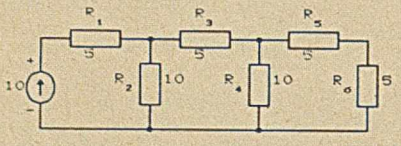
W ten sposób zostaje zrealizowany najprostszy układ generatora składający się z uziemionej pojemności zasilanej ze stałego źródła prądowego. Podobne reguły, dotyczące m.in. zasad użycia wzmacniacza operacyjnego, wtórnika i sprzężenia zwrotnego, umożliwiają syntezę innych rozwiązań generatora, np. integratora Millera lub układu typu *bootstrap*.

W pracach [2, 12] przedstawiono metodę poszukiwania struktur filtrów aktywnych, wychodząc z pewnej struktury uogólnionej. Kolejne topologie powstają przez usuwanie elementów nadmiarowych i zastępowanie pozostałych elementów uogólnionych, elementami rzeczywistymi (R lub C). Uwzględnia się przy tym liczne kryteria podstawowe zapewniające realizowalność transmitancji i potencjalną stabilność, jak również inne kryteria będące funkcją struktury topologicznej – np. sekwencję niezależnego strojenia. Przestrzeń poszukiwań jest zwykle bardzo duża, a skuteczność poszukiwań zależy silnie od przyjętej strategii [2].

Analiza układów

Ważnym elementem w procesie projektowania jest analiza układu. Do wykonania „ręcznej" analizy układu potrzebny jest jego schemat, opis elementów i prawa Kirchhoffa. Próba naśladowania ręcznej analizy doprowadziła do powstania koncepcji Języka Ograniczeń (ang. *Constraint Language*) [18].

Rozważmy opis funkcjonalny rezystora, czyli prawo Ohma – $V = I * R$. Związek ten jest traktowany jako ograniczenie nakładane na trzy zmienne V, I oraz R. Wystarczy teraz znać wartości dowolnych dwóch spośród trzech zmiennych, aby obliczyć trzecią.



Rys. 7. Obwód do przykładu obliczania napięć węzłowych metodą propagacji ograniczeń

Zastosowanie podobnych zasad do obwodu i techniki wnioskowania wstępującego prowadzi do metody nazwanej propagacją ograniczeń [18]. Niech przykładem będzie układ drabinkowy przedstawiony na rys. 7. Do obliczenia napięć węzłowych zastosujemy metodę Guillemina. Zakłada się, że V_{R6} ma wartość e. Dalej przebieg wnioskowania jest następujący:

a) jeżeli $V_{R6} = I_{R6} R_6$, $V_{R6} = e$ i $R_6 = 5$, to $I_{R6} = e/5$

b) jeżeli $I_{R6} - I_{R5} = 0$ i $I_{R6} = e/5$, to $I_{R5} = e/5$

c) jeżeli $V_{R5} = I_{R5} R_5$, $I_{R5} = e/5$ i $R_5 = 5$, to $V_{R5} = e$

d) jeżeli $V_{R6} + V_{R5} - V_{R4} = 0$, $V_{R6} = e$ i $V_{R5} = e$, to $V_{R4} = 2e$

e) jeżeli $V_{R4} = I_{R4} R_4$, $V_{R4} = 2e$ i $R_4 = 10$, to $I_{R4} = e/5$

itd.

f) jeżeli $V_{R1} + V_{R2} - 10 = 0$, $V_{R1} = 4e$ i $V_{R2} = 4e$, to $e = 5/4$

Znając wartość e, można z łatwością wyznaczyć napięcia węzłowe.

Język Ograniczeń zastosowano przy realizacji systemów EL (1975–1976) i SYN (1980), odpowiednio do analizy i syntezy układów. W pracy [17] technikę tę rozszerzono na koncepcję układów równoważnych.

Prace Stallmana i Sussmana [17, 18] oraz oparte na nich prace [9, 19] miały duże znaczenie dla rozwoju nowych koncepcji w realizacji systemów ekspertowych. W teorii obwodów mają one jednak znaczenie dydaktyczne. Dla potrzeb praktycznych, analizę można znacznie sprawniej przeprowadzić przy zastosowaniu metod numerycznych.

Jak już wspomniano na wstępie, do analizy układów stosuje się obecnie symulatory, z których najbardziej popularny jest SPICE [13]. Przy korzystaniu z tych programów powstają czasami trudności z uzyskaniem zbieżności numerycznej; można ich uniknąć, ale trzeba znać przyczyny ich powstawania.

Zadaniem systemu AUSPICE [10] jest asystowanie przy użytkowaniu symulatora SPICE. Podstawowe jego zadania to rozwiązywanie problemów związanych ze zbieżnością algorytmów iteracyjnych oraz szczegółowa analiza danych wejściowych. W przypadku braku zbieżności – dotyczy to głównie analizy stałoprądowej i przejściowej – system doradza lub nawet sam koryguje takie parametry kontrolne procedur numerycznych, jak ABSTOL, RELTOL, TRTOL, VNTOL, ITL1–4 [13]. SPICE dokonuje wprawdzie rutynowej kontroli poprawności danych, ale AUSPICE wyszukuje dodatkowo różne „subtelności", które mogą stwarzać problemy ze zbieżnością. Przykładem może być bardzo duża rezystancja całkowita z jakiegoś węzła do masy, powodująca wystąpienie wartości bliskiej zera na głównej przekątnej macierzy przewodności. Innym przykładem może być bardzo mała pojemność z węzła do masy, powodująca nieuzasadnioną redukcję kroku całkowania. Ta ostatnia sytuacja wynika często z przyjęcia niewłaściwych parametrów modeli elementów.

Wynikiem symulacji pracy układu są wartości liczbowe zestawione w tabele – jest to często wydruk liczący wiele stron. Niektóre symulatory mają możliwość graficznego przedstawienia wyników. Mimo takiego ułatwienia proces ich interpretacji jest zadaniem pracochłonnym, wymaga dużej wiedzy i doświadczenia. Dotyczy to zwłaszcza zjawisk związanych z nieliniowościami oraz oceną stabilności i warunków krytycznych. Dlatego też podjęto próby budowy systemów ekspertowych do interpretacji wyników symulacji [1, 3]. Na obecnym etapie rozwoju podstawowym ich zadaniem jest wyszukiwanie stanów krytycznych – takich jak praca w pobliżu punktu przebicia złącza lub prądów złączowych zbliżonych do wartości maksymalnych.

* * *

Ponad połowa układów ASIC wyprodukowana w firmie AT&T Bell Labs, obok części cyfrowej zawierała też część analogową. Mimo, że obie części wykonywane są na wspólnym podłożu i w tej samej technologii, sposób ich projektowania bardzo różni się.

Układy analogowe składają się zwykle z nieporównanie mniejszej liczby tranzystorów, ale za to problem ich wymiarowania jest bardzo złożony. Wymiary kanału w dużym stopniu decydują o parametrach tranzystorów, a w konsekwencji całego układu. W układach cyfrowych tranzystory są wykorzystywane jako klucze i mają zwykle najmniejsze dopuszczalne przez technologię wymiary. Tylko te elementy, które występują na ścieżkach krytycznych albo pracują na duże obciążenia, mają zdecydowanie inne wymiary.

W układach analogowych takie czynniki, jak parametry procesu technologicznego, elementy pasożytnicze lub dokładność modeli, muszą być uwzględnione już na wysokim poziomie projektowania. W układach cyfrowych tylko ścieżki krytyczne są optymalizowane.

Te oraz inne różnice sprawiły, że do projektowania układów analogowych, systemy ekspertowe zastosowano później i jak dotychczas nie uzyskano takiej ich sprawności, jak w przypadku układów cyfrowych.

LITERATURA

[1] Bergquist S., Sparkes R., QCritic: A rule-based analyser for bipolar analog circuit designs. IEEE 1096 Custom Integrated Circuits Conference. pp. 617–620

[2] Budzisz H.: Searching for TAC filter structures using rule-based system. ECCTD'89, Brighton, U. K., pp. 372–375

[3] Budzisz H.: Interpretacja wyników symulacji programem SPICE przy użyciu systemu reguł. XII KK TOiUE, Rzeszów 1989, pp. 403–408

[4] Carley L. R., Rutenbar R. A.: How to Automate Anolog IC Design. IEEE Spectrum, pp. 26–30, 1988

[5]Degrauwe M. G. et al.: IDAC – an interactive design tool for analog CMOS circuits. IEEE Solid-State Circuits, vol. SC-22, pp. 1106–1116, 1987

[6] DiCataldo G., Nunnari Gm: New trends in circuit design via expert systems. IS CAS'88, pp. 1787–1790
[7] El-Turky F. M.: BLADES – An Expert System for Analog Circuit Design. IEEE IS CAS, pp. 552–555, 1986
[8] Fung A. H., Lee B. W., Sheu B. J.: Self-Reconstructing Technique for Expert System--Based Analog IC Design. IEEE Trans. on CAS, Vol. 36, No. 2, pp. 318–321, 1989
[9] Gams M., Bratko I.: A circuit analysis program that No. 2, pp. 811–815, 1984
[10] Gubian P., Zanella M.: AUSPICE. An expert system to assist simulations with SPICE. IS CAS'88, pp. 889–892
[11] Harjani R., Rutenbar R. A., Carley L. R.: A prototype framework for knowledge--based analog circuit synthesis. Proc. of IEEE Design Automation Conf., pp. 42–49, 1987
[12] Mouly J. C. Neirynck J.: A Prolog program for the exhaustive search of noninteractive tunable biquads. IEEE IS CAS, pp. 544–547, 1986
[13] SPICE User's Manual. MicroSim Corp., 1986
[14] Ressler A. L.: A Circuit Grammar for Operational Amplifier Design. Ph. D. Dissert., MIT 1984
[15] Roylance G. L.: A Simple Model of Circuit Design. MIT Master Thesis in EECS, 1980
[16] Sheu B. J., Fung A. H., Lai Y. N.: A Knowledge Based Approach to Analog IC Design. IEEE Trans. on CAS, Vol. 35, pp. 256–258, 1988
[17] Sussman G. J.: SLICES. At the Boundery between. Analysis and Synthesis. MIT AI Lab., Memo 433, 1977
[18] Sussman G. J., Steele G. L.: CONSTRAINTS – A Language for expressing almoust--hierarchical descriptions. Artificial Intelligence, No. 14, pp. 1–39, 1980
[19] Tanaka T.: Representation and analysis of electrical circuits in a deductive system
[20] Tou J. T., Cheng J. M.: AUTORED: An automated electronic diagram reading machine. Proc. of IEEE Int. Conf. CAD, 1983
[21] Tou J. T., Li W. H., Fan. K. C., Huang C. L.: Knowledge-based approach for the verification of CAD database generated by an automatic schematic capture system. 24th ACM/IEEE Design Automation Conf., pp. 713–720, 1987
[22] Toumazon C., Maleris C. A., Berrah C. M., Cheung P. Y. K.: A methodology for automated generation of analogue integrated circuits. Proc. of ECCTD'89, pp. 624–628, 1989.

Zanim podejmiesz decyzję o imporcie

zaprenumeruj

POLISH TECHNICAL REVIEW

Dwumiesięcznik, wydawany w języku angielskim, szczegółowo informuje o najnowszych osiągnięciach polskiego przemysłu i rodzimej myśli technicznej. Cena prenumeraty rocznej wynosi 110 000 zł (dla zagranicy 24 USD) Wpłaty należy wnosić na konto nr: PBK III Oddział, Warszawa 370015-7490-139-11 Wydawnictwo SIGMA-NOT, Spółka z o.o.
Dodatkowych informacji udziela Dział Kolportażu Wydawnictwa SIGMA-NOT, ul. Bartycka 20, 00–716 Warszawa, tel. 40-30-86 lub 40-00-21 w. 293

abaQ
PRZEDSIĘBIORSTWO ZASTOSOWAŃ ELEKTRONIKI

ul. Obywatelska 128/152, 94–104 Łódź, tel. 86-49-30

oferuje:

- Płytki drukowane jedno- i dwustronne z metalizacją;
- Wykonywanie klisz i projektów płytek drukowanych;
- Programatory EPROM, PROM, 8748 do IBM PC/XT/AT;
- Emulatory EPROM;
- Asemblery i deasemblery skrośne procesorów 8-bitowych;
- Analizatory układów logicznych, 32 kanały, 20 i 40 MHz;
- Falowniki trójfazowe 2,2 kW, zasilane 1- lub 3-fazowe, regulacja kierunku i prędkości do 200%, wejścia sterujące.

0/14/90

SWEDEX UNIVERSAL

SWEDEX UNIWERSAL

Ltd

oferuje:

MODEMY DUPLEKSOWE DO TRANSMISJI DANYCH:

- ☐ pracujące na łączach telefonicznych komutowanych i dzierżawionych, jednoparowych, zgodnie z normami V.21, V.22, V.22 bis CCITT:
  MODEM 1230 P (0–300 b/s asynchr., 600/1200 b/s asynchr./synchr.)
  MODEM 2412 P (0–300 b/s asynchr., 1200/2400 b/s asynchr./synchr.)
  MODEM 2412 H (automatyczny – standard Hayes, 0–300 b/s asynchr., 1200/2400 b/s asynchr./ /synchr.);
- ☐ pracujące na łączach telefonicznych dzierżawionych, dwuparowych: Konwerter KN-9600 (modem typu baseband, 2400 4800 9600 b/s synchr.);
- ☐ pracujące na łączach telefonicznych dzierżawionych, jednoparowych.

Wszystkie modemy posiadają świadectwo homologacji Instytutu Łączności

- ☐ Konwerter RS-232 na pętlę prądową 20 mA (do 19 200 b/s asynchr./synchr., odległość do 2 km).

DO KOMPUTERÓW PC XT/AT:

- ☐ interfejsy pomiarowe IEC 625 (IEEE 488): SU-GPIB (do 10 kB/s), SU-GPIBF (do 500 kB/s);
- ☐ programator SU-PROG pamięci EPROM 2716-27512 i mikrokomputerów jednoukładowych MCS 48, MCS 51;
- ☐ pakiet transmisji synchronicznej BSC: umożliwia pracę IBM PC jako terminala ICL 7181, ICL 7020, IBM 2780, IBM 3780, IBM 3270;
- ☐ oprogramowanie emulacyjne stacji IBM 3780.

DO MERY 9150:

- ☐ Koncentrator Stanowisk Operatorskich KSO-4;
- ☐ Adapter Wyniesionego Stanowiska Operatorskiego ASO z modemami;
- ☐ Adapter Telegraficzny ATG-2.

DO EMC ODRA 1305:

- ☐ Adapter Telekomunikacyjny AT-8/MPX (zastępuje UPD)

TESTERY:

- ☐ KSO (systemy mikroprocesorowe na INTEL 8080 i 8085);
- ☐ V.24 (zgodny z CCITT V.24/V.28 i EIA RS-232 C).

OFERTY – ZAMÓWIENIA – INFORMACJE

SWEDEX UNIVERSAL Ltd
UL. WSPÓLNA 1, 05–75 WESOŁA
k. WARSZAWY
TELEKS 813935 swedex pl,
WARSZAWA TEL. (022) 15-33-65

ZO/238/90

SWEDEX UNIVERSAL

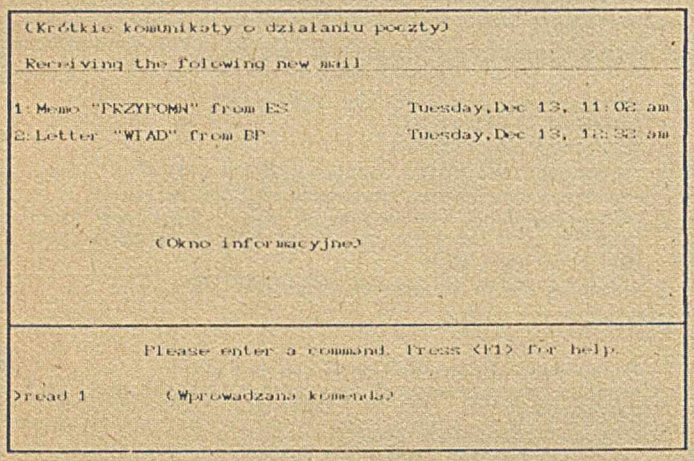
KATARZYNA LANGE
Katedra Informatyki
Uniwersytet Łódzki

Poczta elektroniczna systemu Advanced Netware 286

Poczta elektroniczna to polska nazwa systemu ELECTRONIC MAIL SYSTEM (EMS), stosowanego przez użytkowników sieci ARCNET do tworzenia, wysyłania i składowania korespondencji sieciowej. Tworzenie przesyłek bez potrzeby korzystania z edytora tekstu ułatwia prosty edytor stanowiący integralną część EMS.

Do gromadzenia przesyłek służy specjalny, odrębny katalog, MAIL, którego podkatalogi są skrzynkami pocztowymi poszczególnych użytkowników.

Po uruchomieniu EMS poleceniem MAIL, pojawia się ekran poczty, widziany przez cały czas jej użytkowania. Jest on zbudowany z kilku okienek (rys.). Jedno z okienek stanowią cztery wiersze w dolnej części ekranu (od 22 do 25), związane z wprowadzaniem poleceń. W wierszach 22 i 23 są wyświetlane wiadomości i zgłoszenia dotyczące poleceń (np. reakcja na błędnie wprowadzone polecenie), a w wierszach 24 i 25 wprowadza się polecenia. System zgłasza się znakiem „>" i umożliwia wprowadzanie poleceń o długości do dwóch wierszy.



Ekran komunikacji z użytkownikiem poczty elektronicznej

Podczas wprowadzania poleceń można posługiwać się następującymi klawiszami:

BackSpace, Del	– kasowanie jednego znaku;
← ↑ ↓ →	– przesuwanie kursora;
Esc, Ctrl-C, Ctrl-Break	– kasowanie wprowadzonego polecenia;
F3	– załadowanie wiersza poprzedniego polecenia w celu jego redagowania (nie działa).

Wprowadzanie poleceń kończy się klawiszem *Enter*. Jeśli system nie może zinterpretować polecenia, to umieszcza kursor w miejscu, gdzie była pomyłka, i umożliwia jej poprawienie.

Okno informacyjne zajmuje największą część ekranu. W lewym górnym rogu są wyświetlane informacje – np. o tym, czy dostarczono nową przesyłkę. W oknie informacyjnym można obejrzeć wynik działania poleceń LIST lub READ. Podczas wykonywania polecenia READ na szczycie okienka znajduje się nazwa przesyłki, a niżej jej tekst (zawartość). Tekst można oglądać używając klawiszy ze strzałkami, *PgUp, PgDn, Home* i *End*. Po zakończeniu oglądania zawartości okienka można użyć *F2* lub *Ctrl-Break*, aby powrócić do trybu wprowadzania poleceń. W trakcie wprowadzania polecenia *READ* klawisz *F2* powoduje przerwanie bieżącego wyświetlania i podanie następnej pozycji poczty (jeśli żądano wyświetlania treści większej liczby przesyłek), *Ctrl-Break* natomiast przerywa bieżące wyświetlanie i przywraca tryb wprowadzania poleceń.

Elektroniczna skrzynka pocztowa

Każdy użytkownik (przyłączony do składnicy plików) ma własną skrzynkę pocztową (podkatalog) w katalogu MAIL. Nazwą skrzynki pocztowej jest numer identyfikacyjny użytkownika. Ponieważ EMS ma automatyczny dostęp do skrzynek pocztowych, więc użytkownik nie musi wiedzieć, który podkatalog jest jego. Użytkownik ma wszystkie prawa do swojej skrzynki pocztowej, z wyjątkiem prawa do tworzenia podkatalogów (Parental), a więc może otwierać, czytać, usuwać, przesyłać, zamykać i zachowywać przesyłki do późniejszego użycia.

Zabezpieczenia sieciowe pozwalają innym użytkownikom umieszczać pocztę w czyjejś skrzynce, lecz nie pozwalają im podglądać lub zmieniać zawartości przesyłek, które już tam się znalazły.

Maksymalnie można archiwować 127 przesyłek pocztowych jednocześnie. Teoretycznie każda skrzynka pocztowa jest ograniczona tylko wielkością obszaru pamięci. Aby jednak nie zajmować miejsca na dysku, trzeba okresowo usuwać niepotrzebne przesyłki. Należy też unikać wysyłania dużych przesyłek do licznych użytkowników, ponieważ wtedy każdy z nich otrzyma kopię tej przesyłki do swojej skrzynki. Najlepiej jest w takiej sytuacji wysyłać do każdego użytkownika krótkie informacje (MEMO), gdzie ma szukać tej przesyłki w postaci pliku (trzeba podać katalog).

Każda skrzynka pocztowa zawiera pojedynczy plik (nazywany dalej głównym plikiem poczty), w którym przesyłki są ustawione sekwencyjnie w kolejności odbioru (bo zajmuje to mniej miejsca niż pojedyncze pliki). Nazwa tego pliku to numer identyfikacyjny użytkownika (bez rozszerzenia). Natomiast każda nowa odebrana przesyłka jest umieszczana w pliku o nazwie będącej numerem identyfikacyjnym użytkownika i rozszerzeniu *I0* (dalej nazywanym pomocniczym plikiem poczty). Ten plik pozostaje w skrzynce pocztowej dopóki nie nastąpi ponowne uruchomienie poczty elektronicznej. Wtedy jego zawartość jest dołączana do głównego pliku poczty, a on sam przestaje istnieć. Jeśli zdarzy się, że EMS nie był dawno uruchamiany, a nadeszło kilka przesyłek, to powstanie kilka plików pomocniczych o identycznych nazwach, ale różnych rozszerzeniach – kolejno *I0*, *I1*, *I2* itd.

Jeśli użytkownik skasował jakąś przesyłkę, to zostanie ona zaznaczona w pliku głównym jako przeznaczona do skasowania, ale pozostanie w nim jeszcze do chwili ponownego uruchomienia EMS; dopiero wtedy jest zwalniana część pamięci.

W skrzynce znajdują się również pliki z rozszerzeniem MEM, zawierające notatki (MEMO). Są one tworzone i kasowane przez użytkownika podczas pracy z EMS.

Uruchamianie programu EMS powoduje scalanie głównego pliku poczty, tzn. wymazywanie oznaczonych do skasowania plików i dołączanie nowych przesyłek z pliku pomocniczego poczty.

Przesyłki

Wszystkie typy przesyłek są traktowane jednakowo. Są w taki sam sposób przesyłane i przechowywane w głównym pliku poczty – ich

zawartość, wraz z zapisem skąd pochodzą, kiedy zostały wysłane i jakie mają atrybuty.

W zależności od zawartości, przesyłki klasyfikuje się następująco:
FILE – plik zawierający dane lub program;
DOCUMENT – dokument otrzymywany po przetworzeniu przez edytory tekstu;
LETTER – list używany dla plików tekstowych zawierających wiadomości;
MEMO – notatka używana dla krótkich informacji.

Trzy pierwsze można tworzyć korzystając z edytorów tekstu lub innych programów, przesyłki MEMO natomiast tworzy się z użyciem edytora EMS i są one umieszczane w jego skrzynce pocztowej (długość ich nazwy nie przekracza ośmiu znaków).

Różne typy przesyłek ułatwiają przeglądanie zawartości skrzynki. Rozróżnienia między FILE, DOCUMENT i LETTER dokonuje sam użytkownik, podając przed nazwą przesyłki jej typ. Domyślnie jest przyjmowany typ FILE.

Należy pamiętać o tym, że przesyłce można nadać typ tylko raz i jest on zachowany do końca jej istnienia.

Oprócz typów, do opisu przesyłek używa się również atrybutów. Atrybut FORWARDED opisuje przesyłkę wielokrotnie przesyłaną, tzn. taką, którą sam nadawca już wcześniej od kogoś otrzymał. EXPRESS opisuje pilną przesyłkę, którą przy wysyłaniu oznaczono słowem EXPRESS i o której nadejściu odbiorca jest informowany natychmiast (jeśli aktualnie pracuje w sieci).

Segregacja przesyłek

Przesyłki można segregować według typów, źródła, daty i czasu przesłania lub stosując kombinację tych kryteriów. Segregacji dokonuje się przez podanie opisów – fraz (ang. *search descriptions*) wymaganych przez liczne polecenia EMS.

- Segregacja wg typów

LIST DOCUMENTS AND FORWARDED MEMOS – wypisywanie dokumentów i notatek (MEMO).

- Segregacja wg źródła

MAIL FROM *sender* – MAIL oznacza wszystkie typy poczty, FROM rozpoczyna listę nadawców.

- Segregacja wg daty i czasu

Frazy określające czas i datę zawierają słowa MAILED, SENT, DATED, AFTER, BEFORE, DURING, IN, ON, AT.

Data może być specyfikowana jako YEAR, YEAR i MONTH lub YEAR, MONTH i DAY. Można wybierać przesyłki odebrane przed, w czasie lub po specyfikowanej dacie. Czas może być podawany w formacie standardowym 12-godzinnym lub 24-godzinnym, ze wskazaniem na minuty lub nie.

Przykłady

MAILED AFTER 11 PM DURING 1988
MAILED DURING 1987
MAILED ON OCT 18, BEFORE 5 PM
DATED SEPTEMBER
MAILED AT 4:59 AM ON JANUARY 14, 1987

Słowa ALL, ANYTHING, ANYWHONE itp. są opcjonalne i można je pomijać.

Frazy

Należy być ostrożnym w konstruowaniu fraz. Oddzielenie przesyłek słowami AND, OR lub przecinkiem izoluje je logicznie. Dlatego polecenie OPEN ALL FORWARDED MEMOS AND FILLES powoduje otwarcie wszystkich przesyłek klasy FORWARDED MEMOS i wszystkich przesyłek klasy FILLES.

Polecenie REMOVE ALL EXPRESS MAIL, EXPRESS MEMOS AND FILES kasuje ALL EXPRESS MAIL, ALL EXPRESS MEMOS i ALL FILES.

Wysyłanie wiadomości do grup użytkowników

Gdy użytkownicy są w jakiś sposób określeni, np. posegregowani wg zawodów, funkcji lub wydziałów, to można używać nazw tych grup i ich kombinacji w celu wysłania do nich wiadomości. Można też używać liczby mnogiej, np. PRACOWNICY IN INSTYTUT.

Składnia	**Przykłady**
ALL *tytuł* IN/OF *rodzaj*	ALL SECRETARIES OF SOFTWARE
tytuł IN/OF *rodzaj*	MANAGERS IN ENGINEERING
EVERYONE/ANYONE IN *rodzaj*	ANYONE IN MARKETING
ALL *rodzaj tytuł*	ALL MARKETING SECRETARIES
rodzaj tytuł	SOFTWARE CONSULTANTS

Opis poleceń poczty

Polecenia poczty nie są skomplikowane; do ich zapisania wystarczy podstawowa znajomość języka angielskiego. Format tych poleceń odpowiada formatowi poleceń sieci. Dodatkowo przyjęto następujące konwencje:
– elementy listy muszą być oddzielone przecinkami, słowem AND lub jednym i drugim;
– *path/filename* dotyczy nazwy pliku w systemie DOS, opcjonalnie poprzedzonej pełną lub częściową specyfikacją ścieżki.

Polecenie HELP

Mnemoniki równoważne:
HELP = H = WHAT = WHY = HOW
Powoduje wyświetlenie ekranu z pomocniczą informacją podczas użytkowania poczty.
Składnia:
HELP
HELP *question*
F1 – wywołanie w dowolnej chwili podczas redagowania polecenia. Po wprowadzeniu polecenia HELP odpowiedź można uzyskać dwoma sposobami:
– użyć *F2* i zapytać po angielsku;
– wybrać żądaną opcję, korzystając z klawiszy ze strzałkami, a następnie naciskając *Enter*.
Wyjście z klawiszem *Esc*.

Polecenie CHECK

Mnemoniki równoważne:
CHECK = CHK

Polecenie umożliwia kontrolowanie (podczas pracy z EMS) przesyłek, które zostały odebrane, lecz nie były jeszcze składowane do głównego pliku poczty. Podczas uruchamiania EMS system automatycznie sprawdza, czy nie nadeszła nowa poczta.
Składnia:
CHECK

Polecenie OPEN

Mnemoniki równoważne:
OPEN = 0

Polecenie służy do otwarcia poczty w celu czytania, wyświetlania lub redagowania. Należy podać, jaki rodzaj poczty trzeby otworzyć. Wybrane przesyłki są otwierane i ustawiane w listę; otrzymują identyfikator (liczba z zakresu 1-127), który później jest używany w innych poleceniach.

Po każdym uruchomieniu EMS wszystkie otrzymane ostatnio przesyłki są wyświetlane i otwierane automatycznie.
Uwaga: jednocześnie można otworzyć do 127 przesyłek.
Składnia:
OPEN *search description*

Przykłady:
OPEN ALL MAIL
OPEN ANY MEMOS SEND AFTER DEC 15

Polecenie CLOSE

Mnemoniki równoważne:
CLOSE = C

Polecenie służy do zamknięcia otwartej poczty. Przesyłki pozostają nadal w skrzynce pocztowej.
Składnia:
CLOSE
CLOSE ID *list*
CLOSE *search description*

Przykłady:
CLOSE FILES FROM AZ
CLOSE 1, 2

Polecenie LIST

Mnemoniki równoważne:
LIST = L

Polecenie powoduje wypisanie otwartych lub zeskładowanych przesyłek. Opis przesyłki zawiera typ, atrybuty, źródło i czas wysłania.
Składnia:
LIST
LIST [FILED : ARCHIWED] *search description*
LIST bez parametrów powoduje wypisanie wszystkich otwartych przesyłek.

Przykłady:
LIST FILED MAIL
LIST ALL MEMOS FROM PR

Polecenie READ

Mnemoniki równoważne:
READ = R = SHOW = SCAN

Polecenie powoduje wyświetlenie zawartości wszystkich otwartych przesyłek, określonych przez *search description*. Jeśli podano identyfikatory, to przesyłki zostaną przeczytane w podanej kolejności. Przed użyciem READ poczta musi być otwarta.

Wyświetlanie kolejnych przesyłek następuje po naciśnięciu *F2*, a powrót do trybu wprowadzania poleceń – *Ctrl-Break*.
Składnia:
READ
READ ID list
READ *search description*

Przykłady:
READ MAIL FROM AZ
READ LETTER 7
READ ALL MEMOS

Polecenie PUT

Mnemoniki równoważne:
PUT = P = DUMP = COPY

Polecenie pozwala kopiować dowolną otwartą przesyłkę ze skrzynki pocztowej do pliku w systemie DOS. Kopiowanie do docelowego katalogu jest możliwe tylko wówczas, gdy użytkownik ma do niej prawa *Write, Create, Delete*.
Składnia:
PUT Mail ID TO [*path spec/*] *filename*
PUT MY MEMO *memo* TO [*path spec/*] *filename*

Nazwa *file name* jest nazwą kopii w docelowym katalogu.

Przykłady:
PUT 7 TO SYS: USERS/AA. DAT
PUT MY MEMO AS TO TEST. DAT

Polecenie REMOVE

Mnemoniki równoważne:
REMOVE = REM = ERASE = TRASH

Polecenie powoduje usunięcie otwartych przesyłek z głównego pliku poczty.
Składnia:
REMOVE ID *list*
REMOVE *search description*

Przykłady:
REMOVE ALL MEMOS FROM AA
REMOVE ALL MAIL BEFORE OCTOBER
REMOVE 1

Polecenie SEND

Mnemoniki równoważne:
SEND = S = MAIL

Polecenie służy do wysyłania przesyłki innym użytkownikom lub ich grupom. Do katalogu, w którym znajdują się pliki źródłowe (przesyłki), użytkownik musi mieć prawa *Read, Search* i *Open*.
Składnia:
SEND *source list* TO *target list*
source list: file | document | letter | memo | open mail ID
target list: user name | user group | user group combination

Po zaakceptowaniu polecenia rozpoczyna się przesyłanie; o jego przebiegu nadawca jest informowany na bieżąco. Do przesłania pilnej przesyłki należy używać opcji EXPRESS. Można też wysłać otrzymaną przesyłkę przez jej otwarcie i podanie identyfikatora jako źródła (*source list*) w poleceniu SEND. Przesyłka zostanie wysłana do adresata i otrzyma atrybut FORWARDED.

Przykłady:
SEND LETTER TO AA
SEND DOCUMENT TEKST TO AS
SEND MEMO WIAD TO STUD AND WYKŁ IN INSTYTUT
Lista docelowa jest postaci „STUD" i „WYKŁ IN INSTYTUT", a nie „STUD AND WYKŁ" w „INSTYTUT".

Polecenie LIST USERS AND GROUPS

Mnemoniki równoważne:
LIST USERS AND GROUPS = L

Polecenie służy do wyświetlania użytkowników sieci (LIST USERS), grup (LIST GROUPS), członków grup lub kombinacji grup. Umożliwia to sprawdzenie, kto otrzyma wysyłaną przez użytkownika przesyłkę.
Składnia:
LIST USERS
LIST GROUPS
LIST USERS IN *group name*
LIST [GROUP] *group name*
LIST *group combination*

Przykłady:
LIST USERS
LIST STUD

Polecenie DIRECTORY

Mnemoniki równoważne:
DIRECTORY = D = DIR

Polecenie powoduje wyświetlenie plików w podanym katalogu, pod warunkiem, że użytkownik ma do niego prawo *Search*.
Składnia:
DIRECTORY [*path spec/*] *file name*
Wolno używać znaków * i ?.

Polecenie DIRECTORY bez specyfikacji ścieżki powoduje wyświetlenie plików w bieżącym katalogu.

Przykłady:
DIR E:
DIR *. COM

Polecenie VIEW

Mnemoniki równoważne:
VIEW = V

Polecenie umożliwia oglądanie zawartości każdego pliku (nie związanego z EMS), do którego użytkownik ma prawa Read i Open.
Składnia:
VIEW [file spec/] *file name*
Mogą być specyfikowane pełne lub częściowe ścieżki.

Przykłady:
VIEW AB. TXT
VIEW E: AB. DAT

Polecenie EDIT

Mnemoniki równoważne:
EDIT = E

Polecenie służy do tworzenia i redagowania notatek. Powoduje uruchomienie edytora, którego można użyć do utworzonej nowej notatki lub do redagowania już istniejącej. Maksymalny obszar dla notek wynosi 30 000 znaków.
Składnia:
EDIT [MEMO] *memo name*

Przed redagowaniem istniejące notatki muszą być otwarte poleceniem OPEN. Praca z edytorem jest bardzo prosta. Edytor wstawia słowa do następnego wiersza, gdy zbliżają się do marginesu. Można dowolnie ustawiać kursor i wpisywać znaki w wybranym miejscu. Przejście do następnego wiersza następuje po naciśnięciu klawisza *CR*.

- Przemieszczanie kursora:

← ↑ ↓ → – o jedną pozycję we wskazanym kierunku;
F9 – w lewo do początku wiersza;
F10 – w prawo do końca wiersza;
PgUp – o ekran w górę;
PgDn – o ekran w dół;
Ctrl-Home – do początku notatki,
Ctrl-End – do końca notatki.

- Kasowanie znaków

BackSpace – na lewo od kursora,
Del – znak pod kursorem.

- Kasowanie wiersza

F5 – skasowanie wiersza i przesunięcie w górę tekstu pod nim,
F6 – skasowanie tekstu od pozycji kursora do końca wiersza.

- Przewijanie tekstu

Ctrl-Z – do góry, kursor wędruje razem z tekstem.
Ctrl-W – w dół, kursor wędruje razem z tekstem.

- Odwoływanie redagowania

Ctrl-Break – rezygnacja z redagowanej wersji po twierdzącej odpowiedzi na pytanie „Cancel Memo editing? (Y/N)".

- Kończenie redagowania

F2 – zakończenie redagowania i zapisanie ostatniej wersji notatki na dysku.

Przykłady:
EDIT PRZYKŁAD
EDIT MEMO TEKST

Polecenie LIST MY MEMOS

Mnemoniki równoważne:
LIST MY MEMOS = L MY MEMOS = MLIST = ML

Polecenie służy do wyświetlania utworzonych przez użytkownika notatek.
Składnia:
LIST MY MEMOS [*memo specification*]
Wolno używać znaków * i ?.

Przykłady:
MLIST A??
LIST MY MEMOS S*

Polecenie REMOVE MY MEMOS

Mnemoniki równoważne:
REMOVE MY MEMOS = MREMOVE = MR = REM MY MEMOS = ERASE MY MEMOS = TRASH MY MEMOS

Polecenie powoduje skasowanie utworzonych przez użytkownika notatek. Wolno używać znaków * i ?.
Składnia:
REMOVE MY MEMOS [*memo specification*]

Przykłady:
REMOVE MY MEMOS AS*
MR A??

Polecenie QUIT

Mnemoniki równoważne:
QUIT = Q = BYE = OUT = STOP = END = EXIT = ABORT = SYSTEM

Polecenie powoduje zakończenie pracy z EMS i powrót do systemu DOS.
Składnia:
QUIT

Przykłady:
Q
BYE

LITERATURA

[1] Lewoc J. Sieci komputerowe w praktyce polskiej. Mikroklan nr 7, 1988
[2] NETWARE USER REF FERENCE. PART V ELECTRONIC MAIL. Manual Revision 1.00, Novell Incorporated, January 1986
[3] Sieci komputerowe. Usługi, protokoły, modele. Prace naukowe Centrum Obliczeniowego Politechniki Wrocławskiej. Wrocław, październik 1985
[4] Wrześniak J.: Novell. Mikroklan nr 3, 1988.

Nowe Książki

WYDAWNICTWA NAUKOWO-TECHNICZNE

Jan Bielecki: Turbo Pascal z grafiką dla IBM PC. WNT, Warszawa 1989, wyd. 1, nakład 20 tys. egz., s. 324, ISBN 83-204-0944-6

W książce przedstawiono język programowania mikrokomputerów – Turbo Pascal w jego wersji dla mikrokomputera IBM PC. Dokładnie omówiono wszystkie konstrukcje języka i zestawy podprogramów standardowych, uzupełniając opisy słowne licznymi i szczegółowo skomentowanymi przykładami programów. Wiele miejsca poświęcono problematyce programowania operacji wejścia-wyjścia oraz zastosowaniom języka, jako narzędzia do programowania zagadnień z zakresu grafiki komputerowej na mikrokomputerach rodziny IBM PC. Książka jest przeznaczona dla szerokiego kręgu Czytelników – użytkowników popularnych mikrokomputerów.

Beata Paluszkiewicz, Jerzy Siennicki: WordPerfect 5.0. WNT, Warszawa 1990, wyd. 1, nakład 10 tys. egz., s. 102, ISBN 83-204-1280-3. Seria: Podręczna Pamięć Programisty

W książeczce opisano wersję 5.0 (z maja 1988 r.) edytora WordPerfect firmy WordPerfect Corporation. Podkreślono jego zalety i wady. Konsekwencją olbrzymich możliwości edytora jest to, że w początkowej fazie korzystania z niego użytkownik może mieć kłopoty z zapamiętaniem znaczenia różnych kombinacji klawiszy funkcyjnych.

Witold Sikorski: Programy użytkowe Nortona. WNT, Warszawa 1989, wyd.1., nakład 50 tys. egz., s. 70, ISBN 83-204-1230-7. Seria: Podręczna Pamięć Programisty

Peter Norton jest znanym autorem wielu popularnych programów dla komputerów klasy IBM, pracujących pod kontrolą systemu operacyjnego DOS. Programy te ułatwiają korzystanie z systemu operacyjnego oraz umożliwiają bezpośredni dostęp do informacji zapisanych na dyskach.

III Walny Zjazd PTI

III Walny Zjazd Delegatów Polskiego Towarzystwa Informatycznego odbył się w Warszawie, tradycyjnie w Instytucie Fizyki PAN, 9 czerwca 1990 r. Wzięło w nim udział 97 delegatów reprezentujących 1934 informatyków z całego kraju. Jak zwykle Zjazd rozpoczął się od wręczenia medali laureatom Konkursu im. Jerzego Trybulskiego za najefektywniej wdrożony system informatyczny. Tym razem I nagrodę i medal otrzymał zespół ze Słupska: **Witold Bilicki** i **Karol Czechowski** z Zakładu Projektowania i Usług Inwestycyjnych w Słupsku; wyróżnienie w postaci dyplomu otrzymał zespół z Polskich Linii Oceanicznych, w składzie: **Sanisława Gac, Krzysztof Rogoziński, Aleksander Kierkowski** i **Krzysztof Koszewski**.

W sprawozdaniu Zarządu Głównego poinformowano o działalności Towarzystwa w okresie trzech lat mijających od poprzedniego Zjazdu. Zwłaszcza w ostatnim roku odbyło się wiele spotkań Prezesa z przedstawicielami nowych władz (obecny minister przemysłu, T. Syryjczyk, był zresztą w Zarządzie Towarzystwa i jako gość uczestniczył w Zjeździe). Opracowano i przedstawiono władzom propozycje uregulowań prawnych w dziedzinie prawa autorskiego. Wspólnie z Ministrem Edukacji Narodowej opracowano zasady konkursu informatycznego dla młodzieży szkolnej (początek w roku szkolnym 1990–1991). Towarzystwo tradycyjnie organizuje konkurs im. J. Trybulskiego oraz konkurs na najlepsze prace magisterskie (niestety, ten ostatni nie cieszy się dużym zainteresowaniem). Jest wydawany Biuletyn, choć brakuje do niego materiałów. Towarzystwo próbuje dotować wydawanie wybitnych książek informatycznych oraz udziela stypendiów naukowych. Nie udało się doprowadzić do przełamania monopolu NOT i nadawania przez PTI stopni specjalizacyjnych. Ten temat był jeszcze omawiany w dyskusji. Obecna swoboda kształtowania zasad wynagrodzeń w przedsiębiorstwach zmniejsza rolę formalnych, państwowych stopni specjalizacyjnych, które niegdyś uprawniały do obligatoryjnych dodatków do pensji. Rośnie rola stopni, za którymi stałby autorytet Towarzystwa i które stanowiłyby dla ewentualnego pracodawcy gwarancję autentycznych umiejętności zatrudnianego. Takie stopnie można oczywiście przyznawać bez niczyjej zgody, były jednak głosy, że ze względów formalnych i prestiżowych warto doprowadzić do przełamania monopolu także w zakresie formalnych stopni specjalizacyjnych.

W PTI istnieją trzy Oddziały: Dolnośląski, Małopolski i Górnośląski. Zdecydowanym dysonansem jest brak oddziału w Warszawie (na Mazowszu?). Różnie wygląda działalność kół – przykładem sprawnej działalności jest Koło w Gdańsku, które zorganizowało Centralną Bibliotekę PTI. Powstało kilka nowych sekcji (obecnie jest ich 12). Wszystkie ogniwa PTI prowadziły szczególnie intensywnie działalność popularyzatorsko-szkoleniową, której kulminacją na szczeblu centralnym są słynne już, coroczne Szkoły Jesienne w Mrągowie.

Towarzystwo prowadzi ożywioną działalność międzynarodową. Niestety, także i w tej dziedzinie trzeba walczyć z monopolem, tym razem Polskiej Akademii Nauk, sprzeciwiającej się przyjęciu PTI do IFIP. PTI zorganizowało jednak trzy posiedzenia grup roboczych IFIP w Polsce, a także sfinansowało wyjazd na Kongres IFIP informatyków, wygłaszających tam referaty (PAN wysłała tylko delegację oficjalną). Wymieniono także listy o współpracy z kilkoma towarzystwami zagranicznymi. Przy przygotowywaniu stanowiska nt. ochrony prawnej oprogramowania skorzystano z kontaktów z fachowcami z Wielkiej Brytanii, Japonii, Kanady i USA.

Pieniądze na działalność PTI pochodzą w praktyce wyłącznie z działalności gospodarczej – w większości ze szkoleń, kursów, ekspertyz, analiz i ogólnych projektów, ale także z tworzenia oprogramowania. Zastosowanie ostrzejszych kryteriów doboru prac spowodowało spadek liczby umów, przy jednoczesnym zwiększeniu ich przeciętnej wartości.

Część sprawozdania stanowiło stanowisko na temat stanu polskiej informatyki[1], później przyjęte jako uchwała Zjazdu. Wskazuje się w nim na ciągły brak na polskim rynku i w polskich przedsiębiorstwach dużych komputerów, co negatywnie rzutuje na nasze możliwości rozwiązywania środkami informatycznymi naprawdę dużych problemów. Kadrze informatycznej brak doświadczenia w oprogramowywaniu takich zadań (banki, sieci sklepów, duże towarzystwa ubezpieczeniowe, giełdy). Kolejnym podstawowym problemem jet fatalna sytuacja techniczna sieci łączności oraz zasady taryfowe, utrudniające w sumie łączenie posiadanych komputerów w sieci. Prace programistyczne są zwykle prowadzone chaotycznie, bez metodycznego przygotowania i wykorzystania efektywnych narzędzi i generatorów programów. Większe inwestycje w tej dziedzinie utrudnia powszechne bezkarne piractwo, powodujące zalew taniego (faktycznie – darmowego) oprogramowania. Brakuje standardów, ułatwiających przenośność programów i danych. Badania naukowe także są utrudnione przez brak sprzętu o odpowiedniej mocy – pozostają tylko badania teoretyczne, w których Polska odnosi faktyczne sukcesy.

Nie ma warunków do nauczania informatyki w szkołach. Komputer Elwro 800 Junior, popierany przez PTI, był wadliwie produkowany i naprawiany. Nie powstało oprogramowanie dydaktyczne, brakuje nauczycieli – niezbędne jest działanie Ministerstwa Edukacji Narodowej. PTI podtrzymuje swoje stanowisko, że naukę informatyki można wprowadzać tylko w tych szkołach, które są do tego naprawdę dobrze przygotowane.

Obecny rozwój oprogramowania wiąże się przede wszystkim z działalnością małych firm. Przejście do nich znacznej części najlepszych informatyków grozi obniżeniem morale pracowników dużych firm państwowych, bez których na razie nie mogą powstać duże programy użytkowe. Należy stworzyć warunki, aby także te firmy mogły prawidłowo funkcjonować, nie rezygnując z zajmowania się informatyką profesjonalną. Do prawidłowego rozwoju przemysłu informatycznego trzeba stworzyć odpowiednie warunki finansowe, np. ustalając trzyletni okres amortyzacji na komputery służące do produkcji oprogramowania i na oprogramowanie narzędziowe.

PTI występuje przeciwko mitom o istniejącej w Polsce świetnej kadrze fachowców-informatyków, o sukcesach w produkcji oprogramowania standardowego, o sukcesach eksportowych. W rzeczywistości informatykom brak doświadczenia i kontaktu ze światem, programy zwykle pozostają na etapie prototypów, bez dokumentacji, podręczników, technologii pielęgnowania, sieci dystrybucji itp., a eksport urządzeń nie ma żadnego związku z najistotniejszym w rzeczywistości problemem wdrożeń.

Wyznaczając kierunki rozwoju informatyki należy skupić uwagę właśnie na problemie wdrożeń, czyli obsługi konkretnych dziedzin działalności ludzkiej. Trzeba wspomóc przemysł produkujący oprogramowanie i zapewnić mu odpowiednie warunki (prawo autorskie!). Przy produkcji sprzętu należy na razie korzystać z importowanych podzespołów i elementów. Warto natomiast rozważyć masową produkcję konkretnych, specjalizowanych urządzeń, mogących liczyć na zbyt w kraju (np. kas bankowych i sklepowych).

Kształcenie najwyższej klasy specjalistów-informatyków należy skupić w kilku czołowych uczelniach i odpowiednio je wyposażyć. Informatyce bardziej obecnie potrzeba co roku kilkudziesięciu wybitnych specjalistów niż kilku tysięcy przeciętnych. Wybrane ośrodki naukowe powinny mieć również dostęp do zagranicznych komputerów dużej mocy – może to wymagać poczynienia kroków w celu zniesienia prawnych ograniczeń obowiązujących w tej dziedzinie na Zachodzie. Trzeba wreszcie wprowadzić rozsądne standardy w dziedzinie kodów polskich znaków, zasad komputerowej księgowości, a także wprowadzić nowe akty prawne (ochrona praw autorskich, zasady poufności danych, moc prawna wydruków komputerowych, odpowiedzialność producenta oprogramowania za ewentualne wady itp.). Warto także rozważyć zinstytucjonalizowanie uprawnień informatycznych (na wzór

[1] opublikowane w nr 9, 1990 INFORMATYKI

uprawnień projektowych obowiązujących w innych dziedzinach).

Delegaci przyjęli sprawozdanie Zarządu, a po wysłuchaniu Komisji Rewizyjnej i Sądu Koleżeńskiego prawie jednomyślnie udzielili ustępującym władzom absolutorium. Następnie dokonano kilku zmian w Statucie: umożliwiono członkostwo cudzoziemcom (istotne m.in. ze względu na wzajemną współpracę z towarzystwami zagranicznymi), dopuszczono możliwość nadawania osobowości prawnej Oddziałom (ważne przy prowadzeniu działalności gospodarczej) oraz dostosowano niektóre punkty do nowych przepisów prawnych.

Najistotniejszym zadaniem Zjazdu był wybór nowych władz. Prezesem ponownie został prof. **A. Blikle** (nie miał zresztą kontrkandydata), a do Zarządu weszli kol. kol.: **Borek, Cellary, Czarnowski, Fuglewicz, Gizbert-Studnicki,** Huzar, Irlik, Iszkowski, Maniecki, Mazur, Nałęcki, Nowak, Osuchowska, Paśnik, Szyjewski, Turski, Walenta, Waligórski, Zabrodzki. Wybrano także sześcioosobową Komisję Rewizyjną oraz siedmioosobowy Sąd Koleżeński.

W czasie obliczenia wyników wyborów trwała dyskusja programowa. Minister Syryjczyk poinformował o krokach podejmowanych przez administrację państwową. Zwrócił uwagę, że wiele przepisów powinniśmy dostosować do wymagań międzynarodowych, np. wprowadzenie prawnej ochrony oprogramowania stanowi jeden z warunków zniesienia ograniczeń COCOM. Sceptycznie wypowiedział się na temat nadawania państwowych stopni specjalizacyjnych, przychylając się do koncepcji stopni uznawanych dzięki autorytetowi nadającego Towarzystwa.

Profesor Turski zwrócił uwagę zebranych na pogarszające się warunki kształcenia informatyków w Polsce, na upadek dobrych szkół (Instytut Informatyki Uniwersytetu Warszawskiego będzie miał 30 nie obsadzonych etatów) i rozplenienie się złych (tu nazwy nie padły). Najlepsi informatycy wyjeżdżają za granicę, a poważne projekty programistyczne są prowadzone przez amatorów. Pomoc finansowa i szkoleniowa z Zachodu jest często kierowana przypadkowo, albo ma bardzo kontrowersyjną postać (np. szkolenia traktuje się jako najlepszą formę wyławiania kandydatów do pracy za granicą). PTI powinno ograniczyć własną działalność gospodarczą do ekspertyz i konsultacji wysokiej klasy rzeczoznawców. Inne prace powinny być prowadzone przez powołane spółki lub agencje. O kontaktach zagranicznych mówił m.in. kol. Trybulec. Stwierdził, że słabo są znane programy współpracy i wymiany międzynarodowej, np. TEMPUS (kraje EWG). Inni dyskutanci podkreślali, że w obecnych warunkach rośnie rola Towarzystwa jako autorytetu zawodowego, oceniającego poziom profesjonalny polskich informatyków, współorganizującego konkursy i przetargi, formułującego oceny, kształtującego postawy środowiska (np. w sprawie ochrony oprogramowania).

Zjazd postanowił ustalić wpisowe i składkę członkowską w proporcji do średniej krajowej płacy. O konkretnej wysokości Zarząd Główny będzie co roku informował listownie wszystkich członków.

JAROSŁAW DEMINET

Pakiet Turbo Data Base

Do niedawna wśród narzędzi stosowanych przy tworzeniu systemów z dostępem do bazy danych niepodzielnie panował Clipper. Ostatnio coraz częściej stosuje się klasyczne języki programowania, wśród których najbardziej popularnym jest Turbo Pascal. Za tą zmianą narzędzi przemawiają co najmniej trzy argumenty:

- □ rozpowszechnienie oprogramowania sieciowego umożliwiającego dostęp do bazy danych na poziomie systemu operacyjnego (Btrieve firmy Novell);
- □ rozwój języków programowania, w tym możliwość stosowania programowania obiektowego;
- □ pojawienie się bibliotek modułów organizujących komunikację z użytkownikiem i dostęp do bazy danych.

W przewidywaniu takiego kierunku rozwoju oprogramowania został stworzony pakiet narzędzi dla programistów piszących w Turbo Pascalu o nazwie „Turbo Data Base".

W skład pakietu wchodzą:

1. Zestaw programów narzędziowych (między innymi tworzenie makiet programów, projektowanie struktur baz danych, generowanie podpowiedzi).

2. Biblioteka modułów dla Turbo Pascala obejmująca interface użytkownika (edycja danych, menu, okna) oraz dostęp do bazy danych (indeksy w postaci b-drzew, pliki w standardzie dBase III+, transakcje).

3. Program umożliwiający łatwy dostęp do bazy danych (edycja tabel, wydruki tabulogramów, obliczenia itd...).

4. Przykłady i dokumentacja

W całości oprogramowania pakietu zachowano jednolity standard komunikacyjny (w tym polski alfabet w kodzie Mazowii).

Dostęp do bazy danych za pomocą programów pakietu jest łatwy nawet dla początkującego użytkownika. Istnieje możliwość budowy przejrzystego i niezależnego od programów podsystemu podpowiedzi (Help).

Stosowanie programów narzędziowych TDB wymusza zachowanie pewnych – zgodnych z teorią – zasad już na etapie projektowania systemu. Pozwala to uniknąć większych problemów przy jego modyfikacji i rozbudowie.

Duży nacisk położono na zapewnienie niezawodności oprogramowania i jego odporności na błędy. Istnieje kontrola zgodności pliku z indeksem przy każdym pozycjonowaniu pliku. Zapewniono aktualizację na bieżąco indeksów zadeklarowanych jako stałe. Możliwe jest prowadzenie dziennika, w którym są zapisywane wszystkie zmiany w bazie danych. Umożliwia to tworzenie transakcji i odtworzenie stanu bazy w razie awarii.

Program obsługujący dostęp do bazy jest dostarczany w wersji źródłowej i może być przez użytkownika doskonalony. Dostępne są także źródłowe wersje kilku programów i systemów utworzonych z użyciem pakietu.

TDB to tanie, efektywne, dostosowane do polskich warunków i sprawdzone w praktyce narzędzie wspomagające cały proces tworzenia systemów – od projektu do gotowego produktu.

Trwają prace nad rozszerzeniem pakietu o możliwości stosowania SQL-a oraz wersję pakietu dla TopSpeed Moduli 2.

Cena pakietu: 850 tys. złotych.
Producent: Jerzy Wawro. Pracownia Programistyczna.
37–500 Jarosław, oś. Niepodległości 2/16

0/18/90

Co po czym i kiedy

(O ewaluacji wejścia w shellu Bourne'a)

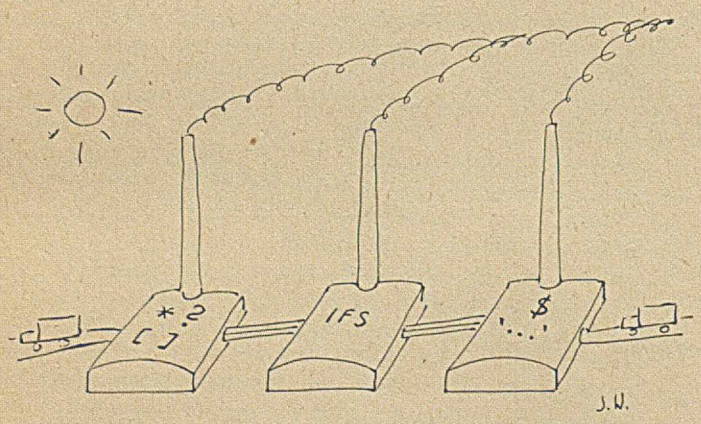
Shell (Bourne'a lub C–shell) jest jednym z ulubionych narzędzi zaawansowanego UNIX-owca. Shell to jednocześnie interpretator poleceń, język programowania i makrogenerator. Bardzo zwięzły skrypt shellowy może wykonać bardzo złożone zadanie. Czasami pisanie takich skryptów jest przedmiotem swoistego współzawodnictwa wśród programistów.

Połączenie mechanizmów wywodzących się z dość różnych dziedzin programowania nie stworzyło jednak ideowo spójnego narzędzia, bo w bardziej złożonych przypadkach mają kłopoty z shellem nawet względnie doświadczeni użytkownicy – dotyczy to zwłaszcza ewaluacji tekstu wejściowego, tzn. mechanizmów kolejnych transformacji tego tekstu. Bywa gorzej: pojawiają się podręczniki (w tym niemieckie, więc wydawałoby się z natury rzeczy solidne) opisujące te mechanizmy w sposób mętny lub wręcz bałamutny. Ponieważ swego czasu zmuszony byłem dość dokładnie przestudiować te zagadnienia, stąd niniejsza próba wyjaśnienia przynajmniej zasadniczych wątpliwości, jakie tu powstają.

Oczywiście, tak krótki artykuł nie wyjaśni natychmiast wszystkich możliwych wątpliwości, ale użytkownik UNIX-a zawsze może podejść do terminala i wykonać test – *experimentum crucis*, który przynajmniej obali jedno z wykluczających się przypuszczeń. Do takiej metody postępowania Czytelników gorąco zachęcam, a poniższe uwagi, mam nadzieję, konstruowanie odpowiednich testów ułatwią.

Kolejność transformacji

Przypomnijmy najpierw kolejność, w jakiej generowane jest polecenie proste. W wyniku rozbioru gramatycznego (analizy składniowej) mamy rozdzielone kolejne słowa polecenia prostego. W ogólnym przypadku jest to jeszcze daleko nieostateczna postać tego polecenia. Jego słowa zostaną poddane procesowi kilku różnych transformacji. Są to:
1) zastępowanie parametrów i poleceń (ang. *parameter and command substitution*);
2) interpretacja odstępów (ang. *blank interpretation*);
3) generowanie nazw plików.



Transformacje następują dokładnie w takiej kolejności i każda tylko raz. Jako przykład rozpatrzmy polecenie trochę sztuczne, ale dobrze ilustrujące te trzy kroki. Polecenie źródłowe ma postać:

```
ls $PATH `pwd`/ slib/menu * > $TEMP
```

Istotne dla nas zmienne shellowe mają wartości:

```
IFS = ':'
PATH = '/bin:/usr/bin:.'
TEMP = '/usr/jasiek/jtmp1'
```

Rozbiór gramatyczny. Nasze polecenie zostanie podzielone na następujące słowa:

```
ls
$PATH
`pwd`
/slib/menu*
>
$TEMP
```

W rozbiorze gramatycznym separatorami słów są wyłącznie znaki spacji i tabulatora. Słowa zaczynające się od znaków specjalnych (np. od >) nie muszą być poprzedzone separatorami.

Zastępowanie parametrów i poleceń. W tym kroku uwzględnia się

1) parametry tzn. zmienne shellowe, parametry pozycyjne i parametry specjalne oraz
2) polecenia ujęte w akcenty (`...`).

Pierwsze są zastępowane swoimi wartościami, drugie – tekstami wyjściowymi produkowanymi przez wskazane polecenia. Po wykonaniu tego kroku otrzymujemy więc:

```
ls
/bin:/usr/bin:.
/usr/jasiek
/slib/menu*
>
/usr/jasiek/jtmp1
```

Interpretacja odstępów. W tym kroku znaki zmiennej **IFS** stają się dodatkowymi separatorami słów. Słowa, które takie znaki zawierają, zostają podzielone odpowiednio na słowa składowe.

```
ls
/bin
/usr/bin
.
/usr/jasiek
/slib/menu*
>
/usr/jasiek/jtpm1
```

Generowanie nazw plików. Uwzględniane są znaki *wzorców nazw plikowych* tzn. * ? [ ]. Zakładając, że istnieją nazwy odpowiadające wzorcowi **/slib/menu***, otrzymujemy np.

```
ls
/bin
/usr/bin
.
/usr/jasiek
/slib/menu1
/slib/menu11
>
/usr/jasiek/jtmp1
```

Wykona się więc ostatecznie polecenie

```
ls /bin/usr/bin . /usr/jasiek /slib/menul /slib/menu11
```

ze skierowaniem wyniku do **/usr/jasiek/jtmp1.**

Proste? W zasadzie tak, ale są jeszcze dodatkowe mechanizmy i ograniczenia, które ten w miarę prosty obraz czynią nieco złożonym. Na przykład, niektóre składniki polecenia podlegają tylko pierwszemu krokowi przetwarzania: dotyczy to skierowania we-wy i podstawień do zmiennych. Jeszcze bardziej komplikują sprawę tzw. *cytowania*. Są one następujące:

'...' – apostrofy: pozwalają włączyć do słowa dowolną sekwencję znaków, które inaczej potraktowane byłyby np. jako separatory składniowe;
\ – ukośnik (ang. *backslash*): działa podobnie jak apostrofy, ale odnosi się tylko do pojedynczego znaku;
"..." – cudzysłowy: działają jak apostrofy, ale nie zabraniają wykonania ewentualnych wewnętrznych zastąpień.

Przejrzymy teraz jeszcze raz wszystkie kroki przetwarzania z uwzględnieniem tego co może je skomplikować.

Rozbiór gramatyczny

W rozbiorze gramatycznym nie są uwzględniane znaki cytowane, a ściślej mówiąc są traktowane jako części słów, nawet jeśli są to separatory lub znaki specjalne. Na przykład, dla polecenia sortowania

sort ⟨aa⟩bb

wejściem standardowym jest plik **aa**, zaś wyjściem plik **bb**. Natomiast dla polecenia

sort ⟨aa'⟩bb'

wejściem standardowym jest plik o dość dziwnej nazwie **aa⟩bb**, zaś wyjście jest dziedziczone po wywołującym shellu.

Zastępowanie parametrów i poleceń

Nie każdy składnik słowa mający postać parametru lub tekstu ujętego w akcenty podlega zastąpieniu. Zastępowanie nie dotyczy składników cytowanych za pomocą apostrofów lub ukośników. Zastępowanie działa natomiast wewnątrz cudzysłowów.

Dla tekstu poleceń wewnątrz akcentów działa normalne przetwarzanie poleceń. Przetwarzanie wykonuje nowa kopia shella ze środowiskiem, dokładnie takim jakie dziedziczą zwykłe polecenia.

Zastępowanie jest jedynym krokiem wykonywanym dla specyfikacji wejść–wyjść i w podstawieniach zmiennych. Na przykład polecenia

AA=*
BB=$AA

nadają obu zmiennym AA i BB wartość *, zaś polecenie

ls⟩*

kieruje wejście programu **ls** do pliku o nazwie *. Ani w jednym ani w drugim przypadku gwiazdki nie są już dalej rozwijane.

Interpretacja odstępów

W tym kroku, jak również w następnym, istotne jest w jaki sposób powstał znak do ewentualnej interpretacji. **Interpretowane są znaki podane bez cytowania i znaki pochodzące z zastępowania parametrów i poleceń. Nie są interpretowane znaki cytowane za pomocą apostrofów, ukośnika i cudzysłowów.** Tych ostatnich dotyczą pewne dodatkowe uwagi, ale o tym za chwilę.

Jeżeli np. **IFS** i **PATH** mają wartości jak poprzednio, następujące słowo

varia:$PATH:'::::'

po interpretacji odstępów stanie się sekwencją słów:

varia
/bin
/usr/bin
.
::::

Aby to wszystko było bardziej czytelne, można zastosować następującą technikę: w tekście uzyskanym z kroku zastępowania podkreślamy znaki cytowane i pozostawiamy je ocechowane w ten sposób do końca ewaluacji. Znaki służące do cytowania pomijamy. **Znaki cechowane nie są interpretowane ani w kroku interpretacji odstępów, ani w kroku generacji nazw.** Uwzględnia się tylko znaki niecechowane – nazywane dalej *czystymi*. Po kroku zastępowania otrzymamy więc następująco ocechowane słowo:

varia:/bin:/usr/bin:::::

i teraz jest widoczne, które znaki są separatorami, a które znakami słowa.

I jeszcze pewna uwaga dotycząca zastąpień wewnątrz cudzysłowów. Symbol $ ⓐ oznacza, jak pamiętają zapewne użytkownicy shella, listę parametrów pozycyjnych porozdzielanych spacjami. Te spacje rozdzielające pozostają *czyste*, nawet jeśli symbol $ ⓐ jest rozwijany wewnątrz cudzysłowów. Natomiast podobny symbol $* rozwija się wewnątrz cudzysłowów ze spacjami rozdzielającymi cechowanymi, tak jak wszystkie inne znaki. Jeżeli parametrom pozycyjnym nadamy wartości za pomocą polecenia jak niżej:

set aa bb cc

wtedy słowo

"$* $ ⓐ "

rozwinie się na sekwencję następująco cechowanych znaków:

aa bb cc aa bb cc

Jeżeli spacja jest znakiem zawartym w **IFS**, ostatnie dwie spacje w powyższym słowie (i tylko ostatnie dwie) zostaną w kroku interpretacji odstępów potraktowane jako separatory.

I jeszcze jedna uwaga, wyjaśniająca dość częste nieporozumienie dotyczące interpretacji odstępów. **Znaki IFS nie mają nic wspólnego z rozbiorem gramatycznym, w sensie rozumianym w tym artykule. W rozbiorze gramatycznym separatorami słów są zawsze i wyłącznie znaki spacji i tabulatora, niezależnie od zawartości IFS.**

Generowanie nazw plików

Tutaj, jak w poprzednim kroku, uwzględnia się tylko *czyste* znaki. Cechowane gwiazdki, pytajniki itd. są traktowane jak jawne znaki nazw plikowych. Na przykład polecenie listowania

ls '?*'

odnosi się do pliku o jawnej nazwie ?*.

Należy tu jeszcze zaznaczyć, że wzorce, które nie dają się rozwinąć, pozostają nierozwinięte bez żadnego komunikatu ze strony shella. Na przykład, jeśli w bieżącym skorowidzu nie ma żadnych plików, polecenie

echo*

wypisze po prostu gwiazdkę bez żadnego dodatkowego komunikatu.

dokończenie na s. 42

„Wszelkie podobieństwo do prawdziwych systemów operacyjnych, żyjących bądź zmarłych, jest czysto przypadkowe."
(z dyskusji o POSIX-ie)

Systemy otwarte i ich wrogowie, czyli co dalej z UNIX-em (3)

Czy grozi nam standardowy UNIX?

Prace nad standaryzacją UNIX-a mają dość długą historię. Rozpoczęte prawie dziesięć lat temu – gdy w ślad za pierwszą schizmą w UNIX-ie pojawiły się trudności z przenoszeniem oprogramowania – wigoru i rozmachu nabrały dopiero w ostatnich latach, kiedy właściwie wszystkie działania *stricte* standaryzacyjne skupiły się w IEEE, w ramach przedsięwzięcia znanego pod kryptonimem **POSIX.** Pozostała część tego opracowania jest niemal w całości poświęcona szczegółom POSIX-owego programu prac standardyzacyjnych.

POSIX *(Portable Operating System Interface for Computer Environments)*, którego pierwotnym celem był standard funkcji systemowych UNIX-a, z czasem przeistoczył się w próbę standaryzacji niemal wszystkiego, co dotyczy systemów otwartych. Prace POSIX-owych grup roboczych potrwają z pewnością jeszcze kilka lat, do tej pory zatwierdzono dopiero jeden spośród kilkunastu planowanych standardów – standard sprzęgu systemowego. Gdy UNIX-owych pierwowzorów nie ma, albo gdy nie są wystarczająco uniwersalne, POSIX-owcy podejmują prace badawcze i proponują zupełnie nowe rozwiązania: w ten sposób zamiast *standardu UNIX-a* otrzymamy kiedyś – w trudnej do przewidzenia przyszłości – *standard POSIX-a.*

Inny rodzaj działalności prowadzi **X/Open**, organizacja założona kilka lat temu przez pięć europejskich firm komputerowych (obecnie ma wielokrotnie więcej członków, z równowagą wpływów *UNIX International* i *OSF*), która formalnie nie jest organizacją standaryzującą, choć bywa za taką uważana. Próbując wypracować coś w rodzaju para-standardu środowiska dla aplikacji, X/Open okresowo publikuje *X/Open Portability Guide* (XPG) – wielotomowy dokument (w ostatnim, trzecim wydaniu ponad dwa tysiące stron) o statusie zestawu zaleceń. Ponieważ X/Open zamierza utrzymywać XPG w zgodności z pojawiającymi się standardami, kolejne wydania XPG mogą istotnie różnić się między sobą – nie tylko objętością. W dziedzinach, w których formalne standardy nie istnieją, X/Open próbuje tymczasowo wypełniać luki odwołując się do standardów faktycznych, lub – w ostateczności – proponując własne rozwiązania. Na XPG można patrzeć jak na zbiór swego rodzaju „ruchomych odsyłaczy" do innych dokumentów: dokumenty wskazywane przez odsyłacz mogą się zmieniać z wydania na wydanie, lecz tylko do momentu, gdy pojawi się zatwierdzony standard.

Ze względów prawnych jedynym dokumentem definiującym UNIX wciąż pozostaje **System V Interface Definition** (SVID), opracowanie publikowane przez AT&T (w wydaniu z 1989 roku około tysiąca stron) – formalnie także nie standard. Zapewne podobną definicję otrzyma również OSF/1. Jak dotąd SVID i inne standardy *de facto* deklarują zgodność z pierwszym POSIX-owym standardem *de iure* i jest nadzieja, że ten obyczaj się utrzyma.

POSIX czyli standaryzacja totalna

Patrząc historycznie, działania POSIX-a są bezpośrednią kontynuacją pierwszych prac nad standaryzacją UNIX-a, zapoczątkowanych w 1981 roku przez **/usr/ group** (największe amerykańskie stowarzyszenie użykowników UNIX-a, obecnie UniForum). W 1985 roku patronat nad nieformalnymi do tej pory działaniami standaryzacyjnymi przejęło IEEE, tworząc długofalowy program prac – znany odtąd jako POSIX (w IEEE identyfikowany numerem 1003) – i powołując kilka grup roboczych.

Od tej pory POSIX-owe standardy opracowuje się wedle przyjętego w IEEE trybu, który w skrócie przedstawia się następująco:
1) grupa robocza tworzy kolejne wersje proponowanego standardu;
2) wersję finalną poddaje się tzw. balotażowi (rodzaj szczegółowego referendum);
3) w przypadku pozytywnego wyniku balotażu propozycja grupy staje się standardem IEEE (po zatwierdzeniu), w przeciwnym przypadku wraca do poprawek.

Standard IEEE ma z kolei duże szanse stać się standardem ANSI, a w końcu – po przyjęciu przez ISO – standardem międzynarodowym.

W 1988 roku zatwierdzono i opublikowano pierwszy z serii planowanych POSIX-owych standardów: **ANSI/IEEE Std 1003.1–1988, IEEE Standard Portable Operating System Interface.** Dokument ten – w środowisku znany jako ***Brzydka Zielona Książka*** (ang. *The Ugly Green Book*), lub wręcz jako **POSIX** – definiuje swego rodzaju minimalny sprzęg systemowy (np. brak wszelkich funkcji systemowych związanych z administrowaniem systemem) oraz nowe, rozszerzone formaty plików archiwalnych typu *tar* i *cpio (dokładniej o Brzydkiej Zielonej Książce napiszemy w INFORMATYCE już wkrótce).*

Z kilkunastu działających obecnie POSIX-owych grup roboczych tylko kilka ma szanse na szybkie zatwierdzenie swoich standardów. Niektóre powołano stosunkowo niedawno, zapewne powstaną następne. Przedstawiony poniżej obraz bieżącego[1] stanu prac każdej z grup powinien dać pewne wyobrażenie o zakresie całego przedsięwzięcia. Stosowana w IEEE numeracja grup z grubsza odpowiada kolejności ich powstawania.

1003.0: Przewodnik po POSIX-ie. Faktycznie ta grupa pełni w IEEE rolę koordynatora i inspiratora większości prac standaryzacyjnych dotyczących systemów otwartych. Jej członkowie to najczęściej kadra kierownicza i czołowi specjaliści przedsiębiorstw prowadzących interesy związane z UNIX-em. To głównie oni zapewniają POSIX-owi reklamę i marketing. Natomiast formalnie jedynym zadaniem tej grupy jest opracowanie szczegółowego przewodnika po POSIX-owych i innych standardach związanych z systemami otwartymi [2].

1003.1: Sprzęg systemowy. Prace tej właśnie grupy zaowocowały pierwszym – i jak dotąd jedynym – z serii POSIX-owych standardów. Obecnie grupa pracuje głównie nad poprawkami i propozycjami rozszerzeń (według sugestii wciąż napływających z pozostałych grup, szczególnie z 1003.8) oraz nad doprowadzeniem *Brzydkiej Zielonej Książki* do postaci wymaganej przez ISO (chodzi tu m.in. o dość kontrowersyjne żądanie uniezależnienia standardu od C, por. 1003.5).

1003.2: Powłoka i narzędzia. Tu niemal od początku prace są prowadzone w dwóch podgrupach. Pierwsza zajmuje się językiem powłoki (najprawdopodobniej będzie to *sh* Bourne'a z elementami *ksh* Korna) i programami usługowymi wykorzystywanymi w skryptach – celem jest właśnie **przenośność skryptów.** W sferze zainteresowania tej podgrupy leżą więc narzędzia używane – z wyjątkiem samej powłoki – w zasadzie nieinterakcyjnie. Z kolei efektem pracy drugiej podgrupy ma być nienormatywne uzupełnienie tego, co stworzy

---

[1]) Pisane latem 1990 roku.

[2]) Notabene w ostatniej wersji tego przewodnika można znaleźć następujące definicje:

System otwarty: system, który implementuje wystarczające *specyfikacje otwarte* dla sprzęgów, usług i formatów, sprawiające, że właściwie skonstruowane oprogramowanie użytkowe może
a) być przenoszone w ramach dużej grupy systemów z minimalnymi zmianami,
b) współdziałać lokalnie lub zdalnie z innymi programami użytkowymi,
c) współpracować z użytkownikami w sposób ułatwiający *przenośność użytkową* (ang. *user portability*, chodzi o to by użytkownik nie napotykał trudności przechodząc od systemu do systemu).

Specyfikacja otwarta: specyfikacja ogólnie dostępna, tworzona w drodze otwartego, publicznego uzgadniania – w procesie umożliwiającym uwzględnianie pojawiających się nowych technologii – i dostosowana do międzynarodowych standardów.

pierwsza o **narzędzia interakcyjne** – w tym przede wszystkim o te, które mają znaczenie w procesie tworzenia, przenoszenia i instalowania oprogramowania (np. *vi, sccs/rcs)*. Podział kompetencji obu podgrup nie jest zbyt ostry, co nie pozostaje bez wpływu na tempo prac (formalnie obie podgrupy są od dawna w trakcie balotażu). Sporo emocji wzbudza propozycja przemianowania *cc* na *c89* (chodzi oczywiście o kompilator ANSI C) – zdaniem niektórych wystarczający dowód na to, że POSIX nie standaryzuje UNIX-a.

1003.3: Atestacja. Ta grupa opracowuje zbiory asercji testowych dla powstających standardów POSIX-a. Po zatwierdzeniu takie zbiory asercji umożliwią zainteresowanym tworzenie programów atestujących – sprawdzających zgodność implementacji ze standardami. Tylko pozornie jest to zadanie niewdzięczne i nudne – żadna inna grupa nie ma tak szczegółowej wiedzy o całości prac standaryzacyjnych. Tempo pracy zależy tu głównie od postępów w pozostałych grupach (na ukończeniu są zbiory asercji dla 1003.1 i 1003.2), a moment ostatecznego zamknięcia prac odsuwa się z powstaniem każdej nowej grupy.

1003.4: Czas rzeczywisty i inne rozszerzenia. Pierwotnie jedynym zadaniem tej grupy miało być opracowanie swego rodzaju dodatku do *Brzydkiej Zielonej Książki*, który standaryzowałby funkcje systemowe potrzebne w zastosowaniach wymagających pracy w czasie rzeczywistym. Z czasem grupa zajęła się również innymi rozszerzeniami, wśród których najwięcej kontrowersji wzbudza mechanizm **wątków** (ang. *threads*). Dyskutowana obecnie propozycja standardu definiuje zestaw dość tradycyjnych środków systemowych związanych z pracą w czasie rzeczywistym, a więc między innymi:

- mechanizmy deterministycznego **szeregowania** procesów (dopuszcza się wiele strategii, w tym również *FIFO*),
- dokładniejsze **czasomierze**, bezwzględne lub względne odliczanie czasu, proces może jednocześnie korzystać z wielu czasomierzy,
- prosty i sprawny mechanizm **semaforów binarnych** (spotykane w niektórych UNIX-ach semafory licznikowe są zbyt uniwersalne, i w związku z tym nieefektywne),
- **pliki** zwarte, możliwość rezerwacji przestrzeni dyskowej, możliwość zapisu bezpośrednio na dysk (z pominięciem buforowania),
- **ryglowanie w pamięci** całości albo części przestrzeni procesu dla ochrony przed wymiataniem lub stronicowaniem,
- **asynchroniczne operacje wejścia–wyjścia** (w klasycznym UNIX-ie procesy czekają na koniec operacji wejścia–wyjścia śpiąc lub drzemiąc),
- sprawny i niezawodny mechanizm **asynchronicznej reakcji na zdarzenia** (takie jak zakończenie asynchronicznej operacji wejścia/wyjścia, czy sygnalizowany przez czasomierz koniec interwału).

Natomiast mechanizm wątków chwilowo znalazł się w dodatku, jako że dyskusje na ten temat wciąż trwają (chodzi m.in. o reakcję wątków na sygnały oraz o zgodność z Adą). Jednak bez względu na losy tej propozycji standardu grupa będzie pracować dalej – najprawdopodobniej w niezależnych podgrupach – nad kolejnymi rozszerzeniami.

1003.5: Powiązania Ady. Zgodnie z UNIX-owymi tradycjami świat języków programowania pierwszych POSIX-owych grup roboczych ograniczał się do C – w kategoriach tego języka zdefiniowano standard sprzęgu systemowego, zapewne podobnie będzie z rozszerzeniami. Wprawdzie pod presją ISO podjęto prace prowadzące do uwolnienia wszelkich standardów POSIX-a od bliższych związków z C (i z jakimkolwiek innym językiem programowania), ale wciąż nie ma zgody co do wyboru języka specyfikacji, nie mówiąc już o sensowności samego zabiegu. Jednakże nawet wówczas niezbędne będą szczegółowe definicje powiązań konkretnych języków programowania ze standardami. W tej sytuacji zdecydowano się powołać grupę, której zadaniem byłoby zdefiniowanie związków Ady z POSIX-owymi standardami, od standardu sprzęgu systemowego poczynając. Nie jest to zadanie łatwe, by wspomnieć tylko trudność włączenia w adowy model współbieżności takich funkcji systemowych, jak *fork* ( ), *exec* ( ) czy *kill* ( ). I w dodatku, żeby to wszystko razem wyglądało jak Ada.

1003.6: Zabezpieczenia. Wbrew obiegowym opiniom dobrze administrowany UNIX stwarza zwykłym użytkownikom stosunkowo bezpieczne środowisko i sam nieźle broni się przed atakami (większość problemów wynika z tego, że UNIX-owe zabezpieczenia najczęściej nie są obligatoryjne). Gorzej natomiast z zastosowaniami, z natury rzeczy wymagającymi zwiększonego bezpieczeństwa – tu rzeczywiście potrzebne są środki, których w UNIX-ie brak. Tak więc zadanie tej grupy polega nie tyle na kodyfikowaniu istniejącego stanu rzeczy, co raczej na próbie wprowadzenia do systemu zupełnie nowych mechanizmów, lecz tak, by to, co powstanie, wciąż jeszcze przypominało UNIX (*szerzej o UNIX-owych wirusach, robakach i o walce z nimi napiszemy w jednym z najbliższych numerów INFORMATYKI)*.

1003.7: Administracja. Ta grupa zajmuje się środkami i metodami administrowania wszelkimi systemami UNIX-owymi (przepraszam: *POSIX*-owymi), w tym także – nowość – sieciowymi (najprostszy przykład zadania dla administratora sieci: założyć użytkownikowi konto na odległej maszynie). Tu również nie ma mowy o kodyfikowaniu istniejących rozwiązań – w administrowaniu UNIX-ami pochodzącymi z różnych źródeł jest zwykle więcej różnic niż podobieństw, nawet w przypadku systemów izolowanych. W tej sytuacji grupa tworzy standard od zera, przyjmując za punkt wyjścia **model obiektowy**: administrator zarządza systemem obiektów należących do różnych klas. Trwa praca nad definiowaniem klas obiektów (np. urządzenie, użytkownik) i ich atrybutów, potem przyjdzie kolej na zestawy funkcji systemowych i poleceń. Standard będzie składać się z części językowo niezależnej (grupa używa ASN.1 – *Abstract Syntax Notation One*) oraz powiązań z C i innymi językami.

1003.8: Pliki w sieciach. Nie ulega wątpliwości, że standaryzacja sieciowych aspektów systemów operacyjnych jest niezbędna – przyznają to nawet UNIX-owi puryści programowo niechętni POSIX-owi. Sprawami sieciowymi w samym POSIX-ie zajmują się trzy grupy, z których najbardziej zaawansowana w pracach jest właśnie ta, standaryzująca mechanizmy tzw. **przezroczystego dostępu do plików** (ang. *transparent file access*). Konkurowały tu dwa podejścia: przekonanie o potrzebie pełnej przezroczystości (plik zdalny widziany dokładnie tak, jak lokalny) oraz silna presja na rzecz „kanonizacji" Sun-owego NFS. W efekcie grupa postanowiła opracować standard dwustopniowy – przezroczystości pełnej (w zakresie całej *Brzydkiej Zielonej Książki*) oraz częściowej (w zakresie ustalonego podzbioru tejże, niesprzecznego z możliwie największą liczbą istniejących protokołów zdalnego dostępu do plików – od standardowego FTAM poczynając).

1003.9: Powiązania Fortranu. Ta grupa stara się osiągnąć dla Fortranu to samo, nad czym – dla swego języka – pracuje grupa Ady. Trudności ma niewiele mniejsze (drobny przykład – jaką nazwę nadać w Fortranie zmiennej *errno*?).

1003.10: Przetwarzanie na superkomputerach. Zastosowania realizowane na superkomputerach mają swoją specyfikę. Typowe programy wymagają całych godzin nieinterakcyjnej pracy. Ze względu na gigantyczne koszty sprzętu przetwarzanie jest tak drogie, że nierzadko budżet kończy się wcześniej niż program. Zdarza się, że spodziewany czas wykonywania się programu przekracza średni czas przedawaryjny. Podstawowy tryb pracy to przetwarzanie wsadowe z połączone z tzw. powstrzymywaniem (ang. *checkpointing*) – techniką umożliwiającą wielokrotne wstrzymywanie i wznawianie długich programów. Grupa superkomputerowa należy do chwilowo nielicznych w POSIX-ie grup, które opracowują **profile**, a nie standardy. (Profil charakteryzuje wymagania użytkownika wobec środowiska i wskazuje na istniejące w danej dziedzinie standardy, ponadto grupa opracowująca profil często inspiruje prace nad nowymi standardami.) Prace nad profilem środowiska dla zastosowań superkomputerowych zmierzają w kierunku określenia zakresu rozszerzeń związanych z przetwarzaniem w trybie wsadowym (mechanizm powstrzymywania, zarządzanie pracami i zasobami, administracja, itd.).

1003.11: Przetwarzanie transakcji. Ta grupa opracowuje profil zastosowań sprowadzających się do przetwarzania transakcji w trybie bezpośrednim. Najprościej mówiąc chodzi tu o zastosowania, w których wielu użytkowników jednocześnie korzysta – często za pośrednictwem sieci – z jednej bazy danych, przy czym użytkownicy są niecierpliwi, a dane cenne. Tradycyjnie ta klasa zastosowań stanowi domenę systemów specjalizowanych (jak

IBM-owski CICS), natomiast klasyczny UNIX nie miał tu zbyt dobrych notowań – głównie za sprawą swego systemu plików. Aby dać UNIX-owi (*POSIX-owi*) szanse i w tej dziedzinie grupa próbuje zdefiniować wymagania stawiane systemowi przez programy tej klasy oraz opracować uniwersalny model przetwarzania transakcji (wzorowany na OSI DTP i X/Open XTP).

1003.12: Komunikacja procesów w sieciach. Powołana w tym roku grupa – druga z sieciowych – zajmuje się standaryzacją tej części sprzęgu systemowego, która dotyczy mechanizmów komunikowania się procesów za pośrednictwem sieci. Prawdopodobnie powstaną dwa warianty sprzęgu – szczegółowy i uproszczony, oba z grubsza na poziomie warstwy transportowej. Przyjęte rozwiązanie ma być swego rodzaju połączeniem mechanizmu gniazdek (ang. *socket*, z 4.3BSD) i XLI.

1003.13: Zdalne wywoływanie procedur. Również tę grupę – trzecią i jak dotąd ostatnią z sieciowych – powołano stosunkowo niedawno. Chwilowo brak dokładniejszych wiadomości o kierunku jej prąc.

1003.14: Przetwarzanie w czasie rzeczywistym. Ta grupa wyodrębniła się ostatnio z 1003.4, w celu opracowania profilu zastosowań wymagających pracy w czasie rzeczywistym.

POSIX a reszta świata

Grupy POSIX-owe nie działają w próżni. Stały wpływ na prace POSIX-a wywierają najnowsze rozwiązania UNIX-owe, w tym również te dopiero zapowiadane. Trzeba liczyć się z wieloma istniejącymi standardami (np. z zaprzyjaźnionym standardem ANSI C) i nie wchodzić w kolizję z grupami standaryzującymi pokrewne zagadnienia (np. sprzęg graficzny, sieci komputerowe). Obszary zainteresowań różnych POSIX-owych grup roboczych też nie są rozłączne. Wszystko to sprawia, że POSIX-owy program standaryzacji będzie z pewnością długim i żmudnym procesem, w którym nawet wcześniej zatwierdzone standardy będą poddawane rewizjom i rozszerzeniom (por. 1003.1).

Wbrew swemu rodowodowi POSIX nie standaryzuje UNIX-a, lecz raczej pewien ideał systemu otwartego, bliski UNIX-owi, lecz wcale z nim nie identyczny. Jakie będą przyszłe losy POSIX-a, trudno przewidzieć. Standardy są towarem i podlegają prawom rynku: gdy brak na nie popytu – idą w zapomnienie. Chwilowo popyt jest, i to duży – władze federalne USA zamierzają kupować wyłącznie oprogramowanie zgodne z POSIX-em (czytaj: z *Brzydką Zieloną Książką*). UNIX, który zawsze będzie POSIX-owi bliższy niż jakikolwiek inny system, niespodziewanie znalazł się w niezwykle uprzywilejowanej pozycji – i ma szansę ją utrzymać.

* * *

Trwa proces odbierania UNIX-a jego autorom i właścicielom. Ale co to właściwie jest UNIX? Przecież nie żywa skamielina. To raczej pewien ogólny pomysł na system operacyjny plus kilka, może kilkanaście pomysłów szczegółowych. Pewna prosta idea wspólna wszystkim wariantom, bez względu na to czy pochodzą z AT&T, z Berkeley czy z OSF. Jeśli tak jest, to z pewnością można twierdzić, że UNIX ma przed sobą ciekawą przyszłość – nawet jeśli za kilka lat zmieni nazwę na POSIX.

JACEK SURMA

*Addenda. Opracowanie jest rozszerzonym zapisem referatu, wygłoszonego na założycielskim spotkaniu Sekcji UNIX-owej PTI w czerwcu 1990 roku. Przygotowując referat korzystałem z informacji pochodzących ze zbyt wielu źródeł (różne numery pism **UNIXWORLD, UNIX Review, login:** i **Byte**, dokumentacja rozmaitych UNIX-ów, podręczniki, monografie, standardy, itp.), by dało się wszystkie rzetelnie odnotować. Innego rodzaju ważnym źródłem była zbiorowa wiedza kolegów z dawnej grupy IPIX-owej z IPI PAN. Wykorzystałem również opinie Glenna Kowacka na temat rywalizacji UNIX International i OSF. Pomysł tytułu zawdzięczam oczywiście Karlowi R. Popperowi oraz wydawnictwu NOW-a, które w 1987 roku wydało jego znakomitą książkę „Społeczeństwo otwarte i jego wrogowie".*

Uwaga! **Redakcja już przyjmuje zamówienia na zamieszczenie ogłoszeń w roku 1991**

LET DRENAŻ=Pomoc

dokończenie z III strony okładki

Dłuższy będzie też okres zwrotu poniesionych nakładów, ale w rezultacie z lepszym pożytkiem dla nas, a również dla nich, jeżeli ich działania są rzeczywiście podyktowane chęcią pomocy, a nie tylko i wyłącznie chęcią rabunkowej eksploatacji.

Podobnie w przypadku kontraktów realizowanych w Polsce konieczne jest zagwarantowanie bezpośredniego udziału w ich realizacji naszych informatyków, którzy po krótkim przeszkoleniu są w stanie podołać powstającym przy tej okazji problemom implementacyjnym. W ten sposób możliwe jest zmniejszenie kosztów pobytu mniej licznych zespołów konsultantów z zagranicy oraz jednocześnie wykształcenie własnej kadry. I nie ma potrzeby bać się takiego żądania, gdyż w tym wypadku to my płacimy za te kontrakty.

Powie ktoś, dlaczego nie wymieniam z nazwy tych Firm, Rządów i Instytucji. Uważam, że nie ma takiej potrzeby, bo po pierwsze znam już kilka takich przypadków w różnych wariantach, a po drugie znacznie ważniejsze jest przekonanie o celowości racjonalnego – z naszego punktu widzenia – postępowania, niż „napiętnowanie" pojedynczego przypadku. Proszę więc traktować te polądy jako opis zjawiska, a nie chęć zablokowania wszelkich działań, tym bardziej, że znajdujemy się naprawdę w trudnym ekonomicznie położeniu i mamy niewielkie pole manewru.

WACŁAW ISZKOWSKI

PS Zbitki poglądów i spostrzeżeń, tutaj prezentowanych, nie są oficjalnym stanowiskiem Redakcji ani żadnej innej organizacji. Ich prezentacja ma służyć pobudzeniu dyskusji na dany temat nawet przez wzburzenie moich ewentualnych adwersarzy.

Co po czym i kiedy

dokończenie ze s. 39

Czy shell Bourne'a jest „internacjonalistyczny"?

Mechanizm cechowania znaków, którego użyłem dla wyjaśnienia reguł ewaluacji, jest faktycznie używany w interpretatorach shella Bourne'a. Do cechowania służy najwyższy bit bajtu przechowującego znak. Nie można więc w shellu Bourne'a korzystać z rozszerzonego kodu ASCII i tym samym ze znaków narodowych. Dotyczy to również C-shella. Produkty te w dotychczasowych realizacjach nie są więc „internacjonalistyczne", aczkolwiek akurat dla tych narzędzi może nie jest to tak bardzo istotne. Nie było tych kłopotów w shellu naszego polskiego IPIX-a, gdyż tam realizacja mechanizmu cechowania była odmienna, ale to już inna historia.

JAN WALASEK

Wychodząc naprzeciw zapotrzebowaniu środowiska, redakcja nasza oferuje łamy INFORMATYKI na bezpłatne zamieszczanie anonsów o imprezach informatycznych organizowanych na terenie kraju.

Zwracamy się do organizatorów konferencji, wystaw, konkursów i sympozjów z prośbą o przesyłanie do redakcji krótkich (pół strony maszynopisu) informacji o planowanych imprezach. Oczekujemy na zgłoszenia.

Czy mógłbyś mnie zabić?

(czyli o utracie kontaktu z rzeczywistością)

Co robić, gdy terminal odmawia posłuszeństwa a uderzanie palcami w klawisze nie przynosi spodziewanych efektów? Postarajmy się w skrócie usystematyzować wiedzę na ten temat i znaleźć środki zaradcze. Zacznijmy od tego, że zgłosiliśmy się w poprawny sposób do systemu i rozpoczęliśmy sesję. Po pewnym czasie pracy przy terminalu znajdujemy się nagle w kłopocie: straciliśmy nad nim kontrolę. Pomijając przyczyny zewnętrzne, takie jak wyciągnięcie przewodu z gniazdka czy awarię systemu, może to być spowodowane:

- zapętleniem lub zawieszeniem programu wywołanego z terminala,
- niepoprawnym zakończeniem programu „ekranowego".

Pierwsza sytuacja jest w UNIX-ie w miarę łatwa do naprawy i prawdopodobnie wszyscy się z nią kiedyś zetknęli. Istnieje jeszcze (w systemach SCO XENIX i UNIX) trzeci powód związany z mechanizmem **mapchan** – wymaga on jednak osobnego omówienia, co obiecuję w jednym z następnych artykułów.

A więc po kolei – zaczniemy od spraw prostszych a wszystkich bardziej „otrzaskanych" przepraszam za nudziarstwo.

Pogoń za zbiegiem

Jedynym sposobem skontaktowania się z procesem zapętlonym lub zawieszonym w oczekiwaniu na jakieś zdarzenie jest mechanizm sygnałów. Dwa sygnały są zwykle związane z klawiaturą: SIGINT i SIGQUIT. Na klawiaturach typu PC sygnałowi SIGINT odpowiada zwykle klawisz **Del**, a drugiemu z nich kombinacja **Ctrl** – \. Standardową reakcją na obydwa sygnały jest awaryjne zakończenie programu (SIGQUIT dodatkowo pozostawia w pliku **core** fotografię procesu w chwili przerwania). Program może oczywiście funkcją **signal( )** przechwycić te sygnały i nie mieć ochoty na zakończenie swojej działalności po ich otrzymaniu. Wówczas jedynym rozwiązaniem pozostaje zwrócić się z dramatycznym tytułowym pytaniem do superużytkownika. Musi on poleceniem **ps** lub **pstat** dowiedzieć się o numer naszego nieszczęśnika a następnie wykonując polecenie **kill – 9 nr_procesu** bezlitośnie go zabić (9 to powszechnie przyjęty numer sygnału SIGKILL, którego proces nie może przechwycić). Całą operację możemy też wykonać sami, gdy mamy dostępny inny terminal, na którym musimy zgłosić się do systemu (inni, „zwykli" użytkownicy naszemu procesowi nie są w stanie wyrządzić krzywdy).

Jedna stąd rada dla programistów: piszcie programy z użyciem funkcji **signal( )** i po otrzymaniu SIGINT i SIGQUIT w sposób uporządkowany kończcie program. Zabicie procesu w sposób opisany wyżej po pierwsze jest postępowaniem awaryjnym, a po drugie pozostawia bałagan. Jednym skutkiem tego bałaganu jest utrata informacji związana z niewłaściwym zamknięciem plików. Drugi skutek może być groźniejszy i o nim za chwilę.

Przywołać terminal do porządku

Wejście-wyjście między terminalem a procesem ulega pośrednim transformacjom w jądrze systemu operacyjnego. Typową taką akcją jest zwrotne wysłanie na terminal znaku otrzymanego z klawiatury (tak zwane **echo**). Klawisz **Enter** produkuje znak **CR (Ctrl-M)**, tymczasem do procesu najczęściej w jego miejsce dociera znak **LF (Ctrl-J)**. W drugą stronę jest podobnie: znak końca wiersza **LF** wysłany z procesu do terminala dociera zwykle jako sekwencja **CR-LF**. Wreszcie wejście z klawiatury terminala może być (i najczęściej jest) buforowane. Dopóki nie wciśniemy klawisza **Enter**, dopóty wpisane znaki nie docierają do procesu. Są to najbardziej istotne z naszego punktu widzenia transformacje. Trzeba jednak wiedzieć, że nie są one jedyne – istnieją transformacje bardziej wyrafinowane. Sterować nimi możemy za pomocą polecenia **stty** lub funkcji systemowej **ioctl ( )**.

Wyobraźmy sobie teraz, że korzystaliśmy właśnie z jakiegoś programu „ekranowego", typu edytora tekstowego lub programu obsługi baz danych. Programy takie, posługujące się pojęciem okien, atrybutów znaków i przemieszczaniem kursora po ekranie ustawiają terminal w stan „surowy" (ang. *raw*), to znaczy rezygnują z niemal wszystkich transformacji dokonywanych przez system operacyjny. Inaczej nie mogłyby prawidłowo działać. Jeśli taki program z jakichś powodów „ulegnie wypadkowi" i w sposób nagły zakończy działanie, terminal nadal pozostaje w trybie **raw**. Jeśli po tej awarii znajdziemy się na powrót w interpretatorze poleceń **sh**, dalsza praca jest niemożliwa, gdyż nie widzimy na ekranie znaków wypisywanych na klawiaturze, a klawisz Enter nie funkcjonuje tak, jak byśmy od niego wymagali. Przyczyny tego są wiadome: brak echa, a klawisz Enter wysyła zamiast LF znak CR, który wcale nie jest rozumiany przez sh jako ogranicznik polecenia. Znak LF uzyskać możemy w inny sposób – wystarczy wcisnąć kombinację **Ctrl-J**. Próbujemy więc uratować sytuację: wciskamy najpierw **Ctrl-J** aby oczyścić bufor wejściowy (mogły w nim zostać jakieś śmieci z naszych wcześniejszych nerwowych prób). Nie przejmujemy się wyzwiskami, którymi nas raczy sh z racji otrzymania takiej porcji bełkotu i wywołujemy (w ciemno – nie widzimy tego na ekranie) program **stty** z parametrem **sane**. Parametr ten oznacza ustawienie na powrót transformacji wejścia-wyjścia do pewnego „rozsądnego" stanu, przyjętego jako domyślny przy rozpoczęciu sesji. Kończymy polecenie ponownym wciśnięciem **Ctrl-J** (nie Enter!).

Po takiej operacji dalsza praca przy terminalu jest zwykle możliwa. Może nam się zdarzyć jeszcze, że:

- cały ekran lub jego fragment otrzymał atrybut, przy którym pracować nie chcemy (np. **reverse** lub **blink**),
- nie widzimy kursora choć domyślamy się gdzie jest.

W takiej sytuacji wywołujemy na chwilę, jeśli uważamy to za bezpieczne, program, który był sprawcą całego nieszczęścia i zaraz z niego wychodzimy. Kończąc działanie w normalny sposób powinien on (chyba że to był nasz własny program) przywrócić terminal do porządku.

Jeśli ten sposób nie pomoże, wyłączamy na chwilę z prądu monitor lub cały terminal. Po ponownym włączeniu i wciśnięciu Enter powinien pojawić się kursor, a niepożądane atrybuty znaków powinny zniknąć (niektóre terminale mają możliwość tak zwanego „miękkiego startu", bez konieczności wyłączania zasilania).

Jeśli i to nic nie da, szukamy naszego zabójcy.

Mina programowa

Dla rozwiania przygnębiającego wrażenia po lekturze tych smutnych rozważań podam na zakończenie przepis na primaaprilisowy żart, który możemy zrobić koledze siadającemu po nas przy terminalu. Oto program w języku **shell**, który wywołujemy tuż przed zakończeniem naszej sesji:

```
1. terminal='tty'
2. (
3. trap " 1
4. sleep 1800
5. parametr=-echo
6. while true; do
7.     stty $parametr
8.     case $parametr in
9.         -*) parametr=echo ;;
10.        *) parametr=-echo ;;
11.    esac
12.    sleep 2
13. done
14. ) <$terminal &
```

Objaśnienia:

wiersz 1: ustalamy nazwę pliku specjalnego odpowiadającego naszemu terminalowi;

wiersz 2 i 14: wykonujemy program jako podproces w tle, ale ze związaniem

dokończenie na s. 45

Argumenty, parametry, środowisko

(w shellu Bourne'a)

Kolejna gawęda o shellu Bourne'a dotyczy przekazywania argumentów procesowi. Sama postać argumentów jest bardzo prosta i obowiązuje wszystkie procesy: zarówno proces shella jak i procesy aplikacyjne. Jednak nieporozumienia w materii przekazywania argumentów procesowi są stosunkowo liczne. Częścią winy należy obarczyć niezbyt szczęśliwie dobraną terminologię i składnię shella Bourne'a, których jednak ze względów historycznych już nie można zmienić. Nie pozostaje więc nic innego, jak postarać się rozgraniczyć precyzyjnie poszczególne terminy i wyjaśnić ewentualne pułapki składniowe. Zacznijmy od omówienia terminologii dotyczącej argumentów procesu.

Proces

otrzymuje *argumenty* pogrupowane w dwie sekwencje:
(1) *argumenty pozycyjne;*
(2) *argumenty kluczowe* albo inaczej *środowisko* (ang. *environment*).

Argumenty są napisami tekstowymi, przekazywanymi za pomocą określonych struktur wskaźnikowych.

Argumenty pozycyjne są dowolnymi napisami, choć oczywiście programy mogą mieć własne wymagania co do ich formy i treści. Oznaczamy je kolejnymi liczbami całkowitymi, poczynając od zera. Argumentem zerowym jest zwykle nazwa programu, ale nie jest to regułą. Niekiedy argumentu zerowego używa się jako opcji wykonania. Na przykład, wywołanie shella z argumentem zerowym zaczynającym się od znaku minusa (–) oznacza żądanie przeczytania i wykonania poleceń z plików **/etc/profile** oraz **$HOME/.profile** przed przejściem do właściwego działania. W ten sposób wywołuje shella inicjator sesji użytkownika.

Argumenty kluczowe mają zawsze postać:

nazwa = wartość

Nazwa poprzedzająca znak = ma postać identyfikatora i jest nazwą argumentu kluczowego, zaś napis po znaku = jest wartością argumentu kluczowego.

Przyjmuje się, że argumenty pozycyjne odnoszą się do zasadniczego działania programu, a argumenty kluczowe do warunków tego działania. Na przykład, w wywołaniu edytora specyfikujemy nazwę redagowanego programu jako argument pozycyjny, a charakterystykę używanego terminala i warunki składowania podajemy w argumentach kluczowych. Uzasadniona jest więc oboczna nazwa listy argumentów kluczowych – *środowisko*. W obrębie tego samego typu przetwarzania używane środowiska różnią się niewiele lub wcale. Na przykład, pracując na tym samym terminalu i redagując różne pliki tym samym edytorem będziemy na ogół używać tego samego środowiska lub, co najwyżej, środowisk niewiele się różniących. Nawet zupełnie różne programy mogą mieć środowiska o wielu elementach wspólnych, np. gdy wykonują operacje ekranowo-zależne, to będą korzystały z identycznych fragmentów środowiska opisujących terminal. W tej sytuacji naturalne jest, że aby uniknąć żmudnego specyfikowania za każdym razem pełnego środowiska, konstruuje się mechanizmy do jego automatycznego kompletowania.

Shell Bourne'a

Zawiera wspomniane mechanizmy, które są ściśle związane z tzw. *parametrami*. Są trzy rodzaje parametrów:
(1) *parametry pozycyjne;*
(2) *parametry specjalne;*
(3) *parametry kluczowe.*

Wartości parametrów oznaczamy symbolami zaczynającymi się od znaku dolara **$**. Wartość parametrów pozycyjnych oznaczamy **$0, $1, $2, $3** itd. Shell inicjuje swoje parametry pozycyjne wartościami otrzymanych w wywołaniu argumentów pozycyjnych, choć z pominięciem niektórych (m.in. opcji, które są odnotowywane gdzie indziej).

Wartość parametrów specjalnych mają symbole **$*, $ @, $ # , $?, $–, $$, $!** i oznaczają różne charakterystyki bieżące shella.

Parametry pozycyjne i specjalne nie mają bezpośredniego wpływu na przekazywane środowisko, więc nie będziemy się nimi zajmować.

Parametry kluczowe oznaczamy identyfikatorami, a wartość parametru kluczowego (zmiennej) oznaczamy jego identyfikatorem poprzedzonym znakiem **$**. Parametry kluczowe nazywamy też *zmiennymi shellowymi* lub – o ile nie powoduje to niejednoznaczności – krótko *zmiennymi*.

Rozpoczynając działanie shell inicjuje zmienne na podstawie odziedziczonego środowiska: dla każdego argumentu kluczowego tworzy parametr kluczowy o nazwie i wartości początkowej takiej samej, jak nazwa i wartość argumentu.

Przypuśćmy, że shell został wywołany ze środowiskiem:

HOME = /usr/jasiek
HZ = 50
PATH = /bin:/usr/bin:.

Po starcie shell utworzy więc trzy zmienne **HOME, HZ** i **PATH** oraz nada im wartości początkowe, jak wyżej. Potem, o ile odpowiednich argumentów nie było w środowisku, utworzy następujące zmienne z wartościami domyślnymi:

IFS, PATH, PS1, PS2.

Używając poleceń złożonych z samych przypisań np.:

HOME = /usr/jasiek/lit BACKUP = Y FREQ = 10

możemy zmieniać wartości istniejących zmiennych i tworzyć nowe. Bieżącą wartość zmiennych możemy obejrzeć wydając polecenie **set** bez argumentów. Wypisana przez **set** lista zmiennych będzie zawierała:

BACKUP, FREQ, HOME, HZ, IFS, PATH, PS1, PS2.

Powiedzmy, że teraz wydamy polecenie wykonania określonego programu, np.

prg vid5

Shell wywoła program **prg** z dwoma argumentami pozycyjnymi:
prg – zerowym
oraz
vid5 – pierwszym.

Program otrzyma również określone środowisko. Podstawą tworzenia środowiska wywołanego programu jest odziedziczone środowisko shella. W naszym przypadku środowiskiem wywołania **prg** będzie właśnie dokładnie odziedziczone środowisko shella:

HOME = /usr/jasiek

HZ=50
PATH=/bin:/usr/bin:.

Czyżby więc ani zmiana wartości zmiennych, ani dodanie nowych zmiennych nie miały wpływu na przekazywane środowisko? Rzeczywiście nie mają wpływu, chyba że zmiennym nadamy atrybut **export.** Nadajmy taki atrybut niektórym zmiennym:

export BACKUP FREQ HOME

Teraz takie samo wołanie programu **prg** powoła proces z innym środowiskiem:

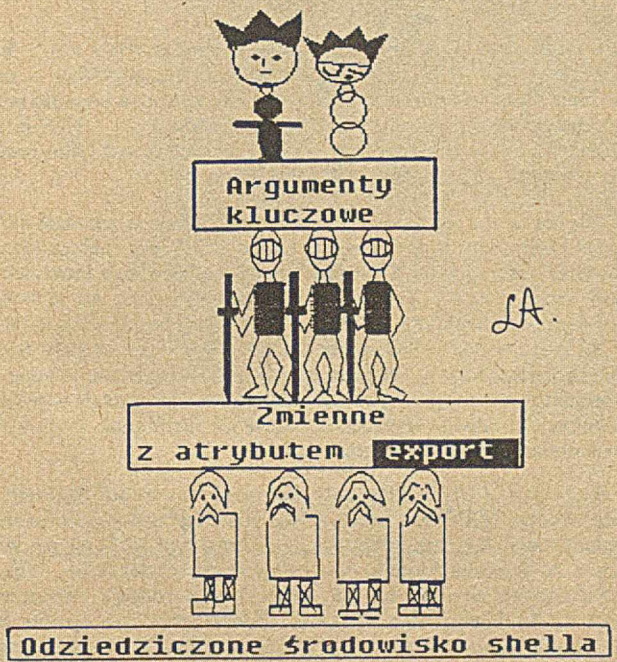
BACKUP=Y
FREQ=10
HOME=/usr/jasiek/lit
HZ=50
PATH=/bin:/usr/bin:.

Jak to powstało? Podstawą pozostaje odziedziczone środowisko shella, ale jego argumenty kluczowe są przebijane bądź uzupełniane przez parametry kluczowe (zmienne) z atrybutem **export.**

Z kolei tak skonstruowane środowisko jest przebijane i uzupełniane przez jawne argumenty kluczowe, o ile takie wystąpią w wierszu polecenia, np.

FREQ=20 prg vid5

Środowisko ostatniego wołania będzie się różniło od poprzedniego tylko argumentem kluczowym **FREQ**, który otrzyma wartość **20**. Jednak zmienna **FREQ** zachowa swoją dotychczasową wartość **10** bez żadnej zmiany!



Rys. LUDMIŁA ANDRUSIEWICZ

Specyfikowanie jawnych argumentów kluczowych nie wpływa ani na zmienne, ani na odziedziczone środowisko shella.

Wyjaśnijmy tu pewne subtelności składniowe. Za pierwszym razem, kiedy użyliśmy zapisu **FREQ=10,** wystąpił on w wierszu polecenia zawierającym tylko analogiczne przypisania i wtedy **FREQ** był symbolem zmiennej. W ostatnim poleceniu zapis **FREQ=20** wystąpił w wierszu wołania programu **prg**, a w takim kontekście **FREQ** jest nazwą argumentu kluczowego.

Na tym jeszcze nie koniec komplikacji z rozróżnieniem argumentów kluczowych i parametrów kluczowych (czyli zmiennych). Weźmy pod uwagę następujące polecenie:

FREQ=30 set –k

Formalnie ma ono postać taką, jak polecenie wywołujące program **prg**. Ale **set** jest tzw. *poleceniem specjalnym*, nie tworzy się dla niego odrębnego procesu, lecz wykonuje w procesie shella i w konsekwencji **FREQ=30** jest w takim kontekście znów przypisaniem do zmiennej shellowej, a nie specyfikacją argumentu kluczowego. Dotyczy to wszystkich poleceń wykonywanych w procesie shella. Jak widać, w shellu Bourne'a nie należy zbytnio sugerować się podobieństwem formalnego zapisu.

Jeżeli nie jesteśmy pewni, czy dobrze skonstruowaliśmy środowisko, możemy wywołać program **env** (w starszych wersjach **printenv**). Wypisuje on po prostu środowisko, jakie otrzymał. Po napisaniu:

env

lub

FREQ=20 env

ujrzymy środowisko, odpowiednio dla przedostatniego i ostatniego wołania programu **prg.**

Zmienne shella możemy usuwać poleceniem specjalnym **unset,** np.

unset BACKUP FREQ

Polecenie to likwiduje nie tylko zmienne shellowe, ale również argumenty kluczowe odziedziczonego środowiska shella. Jest to więc polecenie bardzo mocne w swym likwidatorskim działaniu.

Gdyby **prg** był nie programem, lecz skryptem shellowym, wówczas polecenia tego skryptu wykonywałyby się w takim środowisku shella i z takim stanem zmiennych, jak po wywołaniu:

sh prg vid5

lub odpowiednio

FREQ=20 sh prg vid5

Nie jest to prawdą w odniesieniu do starszych wersji UNIX-a lub XENIX-a, ale ponieważ wersje te szybko wychodzą z użycia, nie będę tych rozwiązań omawiał.

JAN WALASEK

Czy mógłbyś mnie zabić?

dokończenie ze s. 43

wejścia z naszego terminala (plik specjalny musi być stale otwarty);

wiersz 3: ignorujemy sygnał SIGHUP, który dotrze do procesu w momencie zakończenia sesji (inaczej proces zostałby zabity);

wiersz 4: proces śpi pół godziny; w tym czasie my już będziemy daleko, a kolega z zapałem rzuca się w wir pracy;

wiersz 6 i 13: od tego momentu działamy w pętli nieskończonej;

wiersz 7: za pierwszym wywołaniem wyłączamy frajerowi echo; po dwóch sekundach dla zatarcia śladów włączamy je z powrotem i tak dalej;

wiersz 8 do 11: przełącznik włącz echo – wyłącz echo;

Przyjemnej zabawy!

DARIUSZ KUPIECKI

/ usr/bin/mail

Mimika pocztowa

Niektórych początkujących użytkowników poczty elektronicznej niepokoją tajemnicze, z pozoru zupełnie bezsensowne sekwencje znaków, które od czasu do czasu pojawiają się w listach odbieranych tą drogą. Na przykład, po napotkaniu znaków : –) daremnie poszukiwalibyśmy we wcześniejszym tekście otwierającego nawiasu. O co więc chodzi?

Tajemnica wyjaśnia się natychmiast, gdy na taką sekwencję spojrzymy przechylając głowę w lewo i zauważymy coś, co przypomina uśmiechniętą twarz. A więc umieszczając w tekście listu takie trzy znaki nadawca prawdopodobnie chce nam dać do zrozumienia, że trochę żartuje. Miesięcznik UNIXWORLD (4, 1990) przytacza inne często spotykane sekwencje tego typu:

:–0	no nie!
:–v	rozmowa
:–V	krzyk
&–|	płacz
:–||	złość
:–C	nadąsany
:–|	ponury
:–[	przygnębiony
>–<	wściekły
:–/	sceptyczny
:–(	smutny
:–I	obojętny
:–>	sarkastyczny
>:–>	diaboliczny
;–)	z przymrużeniem oka
:–,	uśmieszek
:–D	powiedziane z uśmiechem
:–}	szeroki uśmiech
8–)	uśmiech okularnika
(–:	uśmiech mańkuta

Sekwencje mimiczne są krótkie i nie kosztują wiele, za to z pewnością nadają bardziej ludzki wymiar poczcie, która obywa się bez listonoszy. Używajmy więc tych powszechnie znanych i wymyślajmy własne.

Rzymianie zwykli byli kończyć listy grzecznościową formułką, jakby stworzoną dla poczty elektronicznej:

s.t.v.b.e.e.v.

Tylko kto by to dziś zrozumiał.

(JS)

/książki

The Design of the Unix Operating System

Geopolityka sprawiła, że nasza przeciętna wiedza o wnętrzu UNIX-wciąż jest chaotyczna i wyrywkowa[1]. To prawda, że jest to wiedza dość elitarna, w końcu tylko nieliczni z nas zajmują się oprogramowaniem *stricte* systemowym. Sądzę jednak, że co najmniej niektóre wiadomości o budowie i działaniu jądra przydadzą się także tym, którzy system widzą wyłącznie przez pryzmat funkcji systemowych i nie zamierzają nigdy pisać czegoś równie technicznego, jak choćby własny program obsługi urządzenia. Dlatego warto – w ramach odrabiania bibliograficznych zaległości – przypomnieć o książce, w której o wnętrzu klasycznego UNIX-a znajdziemy najwięcej informacji (i to informacji autorytatywnych – na karcie tytułowej widnieje znaczek AT&T), z pewnością więcej niż w dziesiątkach tomów dokumentacji i podręczników.

Monografia Bacha jest doskonale skonstruowana. Pierwsze dwa rozdziały szkicują architekturę, funkcje i podstawowe mechanizmy jądra systemu – tę część książki polecam zarówno fachowcom, jak i nowicjuszom. Następne osiem rozdziałów – zasadnicza część książki – szczegółowo opisuje struktury danych i algorytmy jądra UNIX-a (w edycji **System V Release 2**, SVR2), odpowiedzialne za realizację kilkudziesięciu najważniejszych funkcji systemowych. Kolejno zapoznajemy się tu z gospodarką buforami, systemem plików, zarządzaniem procesami i pamięcią oraz z obsługą urządzeń zewnętrznych. Tę część książki kończy krótka charakterystyka mechanizmu strumieni (ang. *streams*) z SVR3. Z trzech pozostałych rozdziałów jeden dotyczy komunikacji procesów w SVR2 (IPC) oraz w BSD (gniazdka, ang. *sockets*), a dwa ostatnie traktują o zagadnieniach specyficznych dla wieloprocesorowych i rozproszonych realizacji systemu.

Kilkadziesiąt najważniejszych algorytmów jądra Bach prezentuje w postaci nieco uproszczonych fragmentów autentycznego kodu źródłowego – zabieg, na który może sobie pozwolić tylko autor blisko związany z AT&T. Korzyści dla czytelnika są oczywiste – mamy wrażenie niezwykle bliskiego obcowania z systemem, a ponadto przesuwa to w sferę tzw. powszechnie dostępnej wiedzy o UNIX-ie wiele szczegółów implementacyjnych legalnie znanych dotąd tylko posiadaczom licencjonowanego kodu źródłowego systemu.

Książka Bacha jest pomyślana jako podręcznik: w zasadzie może ją czytać każdy, kto programował w C pod UNIX-em. Zakres i poziom szczegółowości sprawia, że błędem byłoby spodziewać się łatwej lektury, choć może książka jest w sumie przystępniejsza od omawianej tu wcześniej monografii 4.3BSD UNIX-a (por. *INFORMATYKA* nr 9,1990) i bardziej nadaje się dla początkujących. Każdy rozdział kończą ćwiczenia, od elementarnych po rzeczywiście trudne (np. zaprojektować oprogramowanie realizujące tzw. terminale wirtualne w okienkach terminala fizycznego). Radzę ich nie omijać, kryją w sobie wiele dodatkowych wiadomości.

Bach opisuje UNIX klasyczny, bez nowości z ostatnich czterech lat (por. *INFORMATYKA* nr 10,1990). Nie znajdziemy tu informacji o dynamicznym łączeniu bibliotek czy o Sun-owym NFS, nie ma wzmianek o X Window System. Na szczęście dla czytelników, większość UNIX-owych nowości to, jak dotąd, swego rodzaju rozszerzenia i dodatki do klasycznego systemu. Dzięki temu monografia Bacha starzeje się powoli, co w naszej niespokojnej branży raczej wyjątkowe.

JACEK SURMA

W tym numerze po raz trzeci spotykają się Państwo na łamach INFORMATYKI z tematyką UNIX-a. W przyszłym roku zamierzamy kontynuować nasz cykl. Prosimy Państwa o opinie o nowo utworzonym dziale. Będziemy wdzięczni za wszelkie uwagi i sugestie, jak powinniśmy go dalej prowadzić. Oczekujemy listów.

*) ***Maurice J. Bach: The Design of the UNIX Operating System. Prentice-Hall, Inc., 1986 r., s. 471. ISBN 0-13-201799-7***

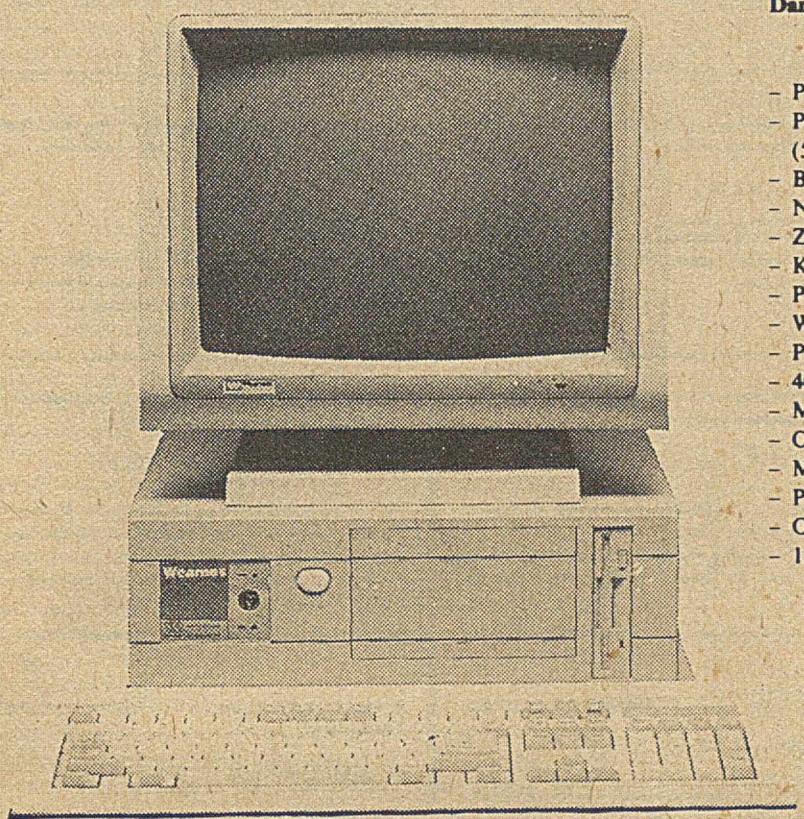
[1] Nawet najbardziej doświadczeni polscy UNIX-owcy z reguły nigdy nie widzieli kodu źródłowego systemu. Twórcy IPIX-a (por. *INFORMATYKA* nr 9,1988) stanowią tu swego rodzaju wyjątek, lecz oni również nie mieli dotąd okazji zobaczyć choćby fragmentów źródeł pierwowzoru swego systemu (IPIX jest z grubsza zgodny z SVR2).

KOMPUTER
Z WBUDOWANĄ PRZYSZŁOŚCIĄ

Komputer zaprojektowany przez czołowego światowego producenta mikrokomputerów – firmę ADVANCED LOGIC RESEARCH i wyprodukowany przez firmę WEARNES TECHNOLOGY.

Komputer, którego możliwości i cena oszałamiają konkurencję.

Komputer, który nie zestarzeje się, dzięki możliwości rozbudowy do architektury 386SX i 486.



Dane techniczne:

SERIA WEARNES BOLDLINE „M"

- Procesor 80286 – 12.5 Mhz
- Pamięć 1 MB RAM, możliwość rozbudowy do 16 MB (5 MB na płycie głównej)
- BIOS Phoenix
- Napęd dyskietek 3,5" 1,44 MB
- Zasilacz 110 Watt
- Klawiatura 101 klawiszy
- Podstawka dla koprocesora matematycznego 80287
- Wbudowany sterownik dyskowy z przeplotem 1:1
- Port szeregowy i równoległy
- 40 MB dysk sztywny
- Możliwość korzystania z EMS 4.0
- Obudowa typu „compact"
- Miejsce na dwa napędy 5,25" o wysokości 1/2
- Podręcznik i dyskietka z programem konfiguracyjnym
- Opcjonalna rozbudowa do 386SX i 486
- 12-miesięczna gwarancja

BOLDLINE COMPUTERS
GRUPA MICOMP-TECH
Biuro Informacji Techniczno-Handlowej
ul. Astrów 7, 40–045 Katowice
telefon i telefaks: 518-628
teleks: 315687 COMP PL

DYSTRYBUTORZY:
PTH „TECHMEX"
43–300 Bielsko-Biała
ul. M. Curie Skłodowskiej 13
tel.: 42-198, 47-555, telefaks: 47-624,
teleks: 35325

EO/1123/89

Fuglewicz P.: Systemy CASE – problemy, techniki, rozwiązania (1)

INFORMATYKA 1990, nr 11–12, s. 3

Przegląd metod i rozwiązań oraz przykłady narzędzi CASE – wspomaganej komputerowo inżynierii oprogramowania.

Fuglewicz P.: CASE systems – problems, techniques and solutions (1)

INFORMATYKA 1990, No. 11–12, p. 3

Review of methods and solutions, as well as tool examples of CASE – the Computer Aided Software Engineering.

Fuglewicz P.: CASE-Systeme – Probleme, Techniken und Lösungen (1)

INFORMATYKA 1990, Nr. 11–12, S. 3

Ein Übersicht von Methoden und Lösungen, sowie Beispiele von Hilfsmitteln der CASE – des coputerunterstützten Softwareingenieurwesens.

Dworakowski K. i inni: Przechowywanie i przetwarzanie danych o złożonych nieregularnych strukturach

INFORMATYKA 1990, nr 11–12, s. 10

Charakterystyka systemu zarządzania hierarchiczną bazą danych GRBASE, przeznaczonego do obsługi danych o nieregularnych strukturach.

Dworakowski et al.: Storaging and processing of data with complex and irregular structures

INFORMATYKA 1990, No. 11–12, p. 10

Characteristics of the GRBASE, a hierarchic database menagement system, which is appropriated for serving of data with irregular structures.

Dworakowski u.a.: Speicherung und Verarbeitung von Daten mit zusammengesetzten, irregulären Strukturen

INFORMATYKA 1990, Nr. 11–12, S. 10

Eine Charakteristik des hierarchischen Datenbanksystems, das für Bedienung von Daten mit irregulären Strukturen bestimmt ist.

Kłopotek M.: Medytor – mówiący edytor języka polskiego

INFORMATYKA 1990, nr 11–12, s. 15

Charakterystyka możliwości funkcjonalnych i przykład stosowania edytora tekstu uwzględniającego specyfikę języka polskiego.

Kłopotek M.: Medytor – a speaking editor of the polish language

INFORMATYKA 1990, No. 11–12, p. 15

Characteristics of functional possibility and application example of the text editor, which allows specific features of the polish language.

Kłopotek M.: Medytor – ein sprechender Editor der polnischen Sprache

INFORMATYKA 1990, Nr. 11–12, S. 15

Eine Charakteristik der Funktionsmöglichkeiten und ein Beispiel von Anwendung des Texteditors, der die Sonderheiten der polnischen Sprache berücksichtigt.

Wrzeszcz Z.: Język Lisp i elementy sztucznej inteligencji (2). Konstruowanie funkcji i programu

INFORMATYKA 1990, nr 11–12, s. 18

Druga część charakterystyki języka Lisp, zawierająca omówienie specjalnych form funkcjonalnych, niezbędnych do tworzenia programu.

Wrzeszcz Z.: Lisp language and AI elements (2). Constructing of functions and program

INFORMATYKA 1990, No. 11–12, p. 18

Second part of characteristics the Lisp language, which includes discussion of special functional forms needed for program building.

Wrzeszcz Z.: Lisp-Sprache und Elemente der künstlichen Intelligenz (2). Konstruierung von Funktionen und Programm

INFORMATYKA 1990, Nr. 11–12, S. 18

Zweiter Teil einer Charakteristik von Lisp-Sprache, der eine Besprechung der unbedingt zur Programmgestaltung nötigen spezifischen Funktionsformen umfasst.

Muszyński M.: Fizyczne struktury baz danych (3). Wielowymiarowe struktury danych

INFORMATYKA 1990, nr 11–12, s. 23

Trzecia część wprowadzenia w dziedzinę fizycznych struktur systemów baz danych, zawierająca omówienie możliwości realizacji dostępu według wielu atrybutów w bazach danych opartych na typowych, jednowymiarowych organizacjach plików danych.

Muszyński M.: Physical structures of database systems (3). Multidimensional data structures

INFORMATYKA 1990, No. 11–12, p. 23

Third part of an introduction to the field of physical database structures, which includes discussion of access realization possibility according to many attributes in databases built on typical, one-dimensional disk file organizations.

Muszyński M.: Physische Strukturen von Datenbanksystemen (3). Mehrdimensionale Datenstrukturen

INFORMATYKA 1990, Nr. 11–12, S. 23

Dritter Teil einer Einführung im Bereich der physischen Strukturen von Datenbanksystemen, der eine Besprechung von Realisierungsmöglichkeiten des Zugriffes nach vielen Attributen in Datenbanken mit typischen, eindimensionalen Plattendateiorganisationen umfasst.

Budzisz H.: Systemy eksportowe w projektowaniu układów analogowych

INFORMATYKA 1990, nr 11–12, s. 27

Charakterystyka istniejących rozwiązań w dziedzinie zastosowania systemów eksportowych do projektowania układów analogowych.

Budzisz H.: Expert systems for designing of analog circuits

INFORMATYKA 1990, No. 11–12, p. 27

Characteristics of solutions in expert system application for designing of analog circuits.

Budzisz H.: Expertensysteme in Projektierung von Analogschaltkreisen

INFORMATYKA 1990, Nr. 11–12, S. 27

Eine Charakteristik von Lösungen im Bereich der Expertensystemanwendung zur Projektierung der Analogschaltkreise.

Lange K.: Poczta elektroniczna systemu Advanced Netware 286

INFORMATYKA 1990, No. 11–12, p. 32

Charakterystyka rozwiązań, zasady działania oraz przykłady stosowania poczty elektronicznej w sieci ARCNET.

Lange K.: Electronic mail of the Advanced Netware 286 system

INFORMATYKA 1990, nr 11–12, p. 32

Characteristics of solutions, operation principles, as well as examples of electronic mail application in the ARCNET Network.

Lange K.: Elektronische Post des Advanced Netware 286-Systems

INFORMATYKA 1990, Nr. 11–12, S. 32

Eine Charakteristik von Lösungen und Arbeitsweise, sowie Anwendungsbeispiele der elektronischen Post im ARCNET-Netz.

LET DRENAŻ=Pomoc

Kolejna zagraniczna Firma zaproponowała pewnej Instytucji w Polsce współpracę w przeszkoleniu grupy informatyków. Podstawowymi elementami tej propozycji było:

- *w drodze konkursu zostanie wybrana grupa młodych informatyków po studiach,*
- *program szkolenia zostanie podzielony na dwie części,*
- *pierwsza część szkolenia odbędzie się w Polsce siłami Instytucji (przez jej wykładowców),*
- *druga część szkolenia odbędzie się w kraju Firmy,*
- *Firma pieniądze na szkolenie uzyska od swojego Rządu z puli pieniędzy oferowanych Polsce w ramach międzyrządowych umów o pomocy,*
- *część tych pieniędzy zostanie spożytkowane przez Firmę na przygotowanie programu szkolenia i jego realizacje w kraju Firmy,*
- *pozostała, mniejsza część pieniędzy, zostanie przekazana Instytucji w postaci sprzętu oraz wynagrodzenia dla wykładowców,*
- *być może część wykładowców z Polski pojedzie na wycieczkę naukową do kraju Firmy.*

Takiej propozycji należy tylko przyklasnąć. Zyskujemy w Polsce dostęp do nowego sprzętu, możliwość podniesienia kwalifikacji oraz kontakty z Firmami zagranicznymi. Wykładowcy w Polsce zyskają możliwość dodatkowego wynagrodzenia, a grupa młodych informatyków uzyska dodatkową wiedzę oraz poduczy się języka. Same plusy, żadnych minusów. Czy, aby na pewno tak jest?

Przyglądając się takiej propozycji dokładniej, zauważamy, że Firma jest firmą wykonującą oprogramowanie, a tylko w niewielkim stopniu zajmuje się edukacją informatyków. Dodatkowo dowiadujemy się, że w kraju Firmy brakuje prawie drugie tyle informatyków ilu jest dotychczas, a jest ich niemało. Następnie, zapoznając się z proponowanym programem szkolenia, konstatujemy, że nasi wykładowcy nie będą mieli dużego problemu z przygotowaniem się do jego prezentacji, wszak na podobnym poziomie prowadzą podobne zajęcia na co dzień. Być może dopiero w kraju Firmy poziom szkolenia będzie wyższy, ale o tym trudno się dowiedzieć czegoś bliższego. Dalej, analizując wielkość nakładów na zakup sprzętu, stwierdzamy, że nie może to być sprzęt znacząco lepszy od tego, który jest już w naszym posiadaniu.

Wszystkie te stwierdzenia przeważnie nie budzą jakoś niepokoju u polskiego partnera z Instytucji. Przecież, nie inwestując nic z naszej strony, dostajemy sprzęt i możliwość rozwoju i to jeszcze bez konieczności zwrotu w przyszłości zainwestowanych w to przedsięwzięcie pieniędzy. A oferowanej nam pomocy nie można się przyglądać zbyt dokładnie i na pewno nie można wybrzydzać. W zasadzie należy się zgodzić z takim punktem widzenia, ale...

Właśnie chodzi o to „ale". Dokładniej analizując te propozycje łatwo można przewidzieć następujący dalszy scenariusz. Przeciętnie najlepsi, pozostający jeszcze w kraju informatycy zyskują możliwość douczenia się w zakresie potrzebnym przez Firmę, łącznie z bliższym poznaniem działalności Firmy oraz jej środowiska. Jednocześnie Firma będzie miała możliwość dokładnego poznania zakresu umiejętności i uzdolnień każdego z douczonych polskich informatyków. Ostatnim elementem tego przedsięwzięcia będzie zaproponowanie wszystkim – lub tylko najlepszym – zatrudnienia w Firmie. Tylko niewielu powróci do kraju. Pozostali swoją pracą szybko z nawiązką spłacą i Firmie, i jej Rządowi pieniądze przeznaczone szumnie na pomoc Polsce. A nam w kraju ubędzie kolejnych kilkunastu zdolnych, przeważnie młodych, ludzi. Jednocześnie, wszyscy już chyba są zorientowani ilu naszych starszych i młodszych kolegów (koleżanek też) opuściło nasz kraj i pracuje na Zachodzie w wielu firmach i uniwersytetach. Kilkunastu spośród nich zrobiło rzeczywistą karierę. Wielu stało się zwykłymi programistami, najczęściej w języku Cobol, pracując jako „dochodzący" modyfikatorzy eksploatowanych od lat systemów informatycznych. Wielu też zasiliło średnią kadrę techniczną w państwach arabskich. Nie wszyscy też mieli okazję znaleźć ciekawą pracę w swoim zawodzie. Nawiasem mówiąc warto byłoby dokonać przeglądu naszych kadr za granicą.

Powstaje więc szereg wątpliwości i pytań. Czy należy spokojnie patrzeć na to, jak kolejne kadry odpływają na Zachód? Czy może należy zabrać informatykom paszporty? Czy istnieje możliwość zatrzymania w kraju wysoko kwalifikowanych kadr, w przypadku, gdy małe są szanse na zaoferowanie im ciekawej pracy i radykalny wzrost ich wynagrodzeń? Czy możliwe jest prowadzenie i zrealizowanie dużych projektów informatyzacji gospodarki wyłącznie siłami firm zagranicznych, w sytuacji gdy nisko opłacane zespoły informatyczne ze strony polskiej mają przejąć te systemy do eksploatacji? Na te pytania powinniśmy sobie odpowiedzieć bardzo szybko, jeżeli chcemy rzeczywistego rozwoju naszego kraju.

Uważam, że istnieje rozwiązanie godzące te rozbieżne zjawiska. Możliwe jest bowiem wynegocjowanie takich kontraktów z firmami zachodnimi, które pozwoliłyby z jednej strony na uzyskanie dostępu do najnowszej, wykorzystywanej u nas w kraju, technologii, a z drugiej zapewnienie polskim informatykom ciekawej i dobrze płatnej pracy tu, a nie na Zachodzie. W przypadku przedsięwzięcia opisanego na początku, możliwe jest wynegocjowanie utworzenia w Polsce oddziału Firmy, w którym Polacy będą realizować kontrakty dostarczane przez Firmę, być może z okresowym pobytem w jej macierzystym kraju. Wymaga to oczywiście znacznie większych inwestycji ze strony Firmy, przeznaczonych głównie na zakup i instalację najnowocześniejszego sprzętu oraz kosztowniejszą obsługę w realizacji projektów.

dokończenie na s. 42

PRZEDSIĘBIORSTWO
ZASTOSOWAŃ INFORMATYKI

meditronik

oferuje:

- systemy mikrokomputerowe
- programy aplikacyjne dla różnych dziedzin gospodarki (na życzenie wysyłamy katalog)
- poszukiwane komponenty elektroniczne
- interfejs do kamery video (opc. CCD) z bogatą biblioteką oprogramowania
- emulator Z80
- tester układów scalonych i pamięci
- programator BPROM
- asynchroniczny procesor komunikacyjny
- konwerter RS-232 – Centronics

instaluje:

- połączenia międzykomputerowe (XT/AT – ODRA/RIAD/IBM)
- systemy sieciowe (NOVELL)
- systemy wielodostępne (SCO Xenix 286, 386, Unix System V)

Jeżeli jesteś autorem oryginalnego programu aplikacyjnego

- skontaktuj się z nami, będziemy pośredniczyć w sprzedaży Twojego programu, dbając o ochronę Twoich praw autorskich!

Nasz adres:

00–194 Warszawa, ul. Dzika 4
tel. (02) 635–22–63, 635–22–64,
fax (02) 635–21–95
tlx 816075 medi pl

EO/605/89

supra

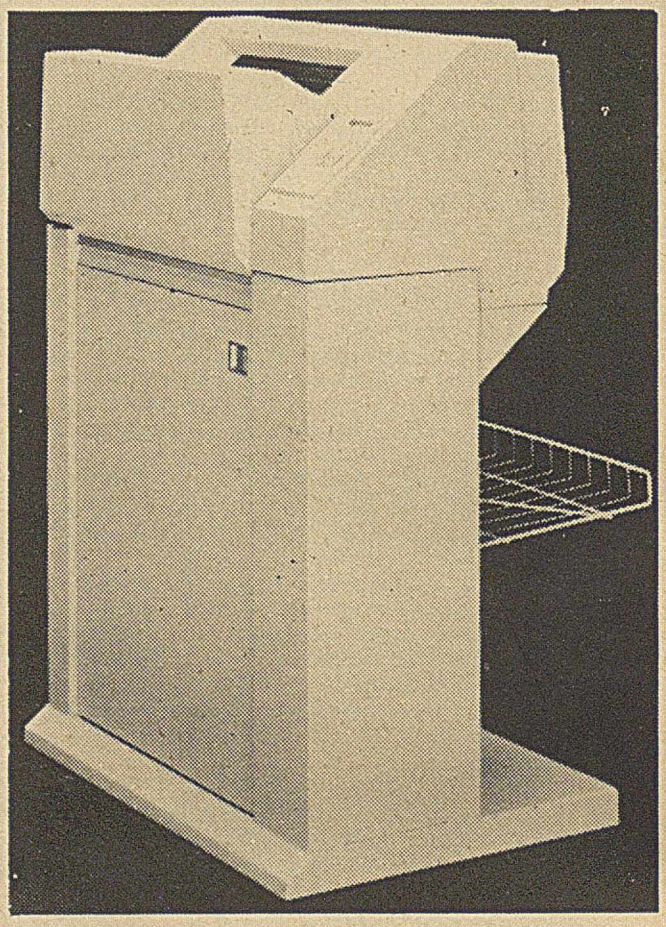
SUPRA SP. Z O.O.
ul. Trybunalska 54, 60-325 POZNAŃ
telefon: 674392, telefaks: 236498
teleks: 412966

OFERUJE ŚWIATOWY BESTSELLER

MANNESMANN
TALLY

DRUKARKĘ MOZAIKOWO-WIERSZOWĄ MT 645

- doskonała jakość druku
- niezawodność 7000 h MTBF
- duża szybkość 450 wierszy/min
- cicha praca
- bufor 27 KB
- liczba kopii 1+5
- pełna grafika 240 (dpi)
- polskie litery (opcja)
- wewnętrzny program testujący
- dwa interfejsy automatycznie przełączane
- podwójny system traktorów



MANNESMANN
TALLY

NAJPRAWDOPODOBNIEJ NAJLEPSZE

DRUKARKI NA ŚWIECIE

AUTORYZOWANY DYSTRYBUTOR DRUKAREK

MANNESMAN TALLY

supra

0/12/90